①

AD-A202 564

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

89 1 17 145

1

AFIT/GE/ENG/88D-39

DESIGN AND IMPLEMENTATION OF A PC-BASED NETWORK SIMULATOR FOR ENGINEERING EDUCATION

THESIS

Ralph D. Puckett Captain, USAF

AFIT/GE/ENG/88D-39

AFIT/GE/ENG/88D-39

DESIGN AND IMPLEMENTATION OF A PC-BASED NETWORK SIMULATOR FOR ENGINEERING EDUCATION

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

Ralph D. Puckett

Captain, USAF

December 1988

Preface

The purpose of this study was to develop a computer aided design (CAD) tool to supplement instruction in computer communications networks. There existed a need for computer aided modeling and analysis of user-defined networks and this work was dedicated to that purpose.

Most of the effort was concentrated on providing a graphical interface between human and machine with the ultimate goal of making the operation of the tool as transparent as possible. Although extensive testing was done during the development, the ultimate test will be whether or not the software helps the student understand concepts presented in textbooks and lecture.

I would like to acknowledge some of the individuals who helped make this work possible. First of all, I'd like to thank my thesis advisor, Capt George, for providing direction, encouragement, and perhaps most of all, a positive attitude. I also wish to thank LTC Garcia for the use of his computer network class in beta testing the software. Thanks go also to my committee members, Capt Davis and Capt Fautheree. Finally, I wish to express my profound gratitude to my wife Haeng Soon for her tolerance, patience, and moral support.

Ralph D. Puckett

Table of Contents

Page

Preface ii

List of Figures iv

Abstract v

I. Introduction 1

II. Review of the Literature 6

III. Network Theory 13

Networks of M/M/1 Queues 13

Flow and Capacity Assignment 17

IV. Software Development 20

Requirements 20

Design 23

Implementation 32

Detailed Development 38

Testing 40

V. Results and Discussion 42

Development Methodology 42

Test Results 42

Recommendations 45

Appendix A: User's Manual 47

Appendix B: Tests 65

Appendix C: Software Evaluation Survey Results 79

Bibliography 81

Vita 82

List of Figures

Figure Page

1. Network Modeling....14
2. Network Parameters (Input)....16
3. Network Parameters (Calculated)....18
4. Data Structure Design....26
5. Structure Chart - High Level Design....27
6. Structure Chart - High Level....35
7. Structure Chart - Network Editor....36
8. Structure Chart - Routing Editor Module....37
9. Structure Chart - Analyzer Module....38
10. Comparison with Comnet II.5....44

AFIT/GE/ENG/88D-39

Abstract

The purpose of this study was to design and implement a simulator to assist students of computer networks. The basic objective was to create a software application that provides rapid feedback on network design decisions. Of particular interest is the packet switched network with data links of various capacity assignments. Another basic objective was to create a graphics interface that eliminated the need to learn a simulation language while still maintaining a powerful and useful product.

The end product was a result of the application of both networking theory as well as software engineering principles with particular attention being paid to reliability and maintainability. With this tool the student can create any network topology simply by pointing and clicking a mouse and entering a few network parameters from the keyboard. The application can be run on a personal computer - an environment which is accessible and fairly well understood.

The design and implementation of the application is presented in this work along with the results of developmental testing. In addition, a sample of fifteen students was chosen to provide the initial beta test and those results are presented. A comprehensive user's manual is included as an appendix. Finally, several recommendations concerning future development are discussed.

DESIGN AND IMPLEMENTATION OF A PC-BASED NETWORK SIMULATOR FOR ENGINEERING EDUCATION

I. Introduction

Overview

The Department of Electrical and Computer Engineering of the Air Force Institute of Technology is involved in an on-going process of building a catalog of software tools that are developed in-house. The fundamental purpose of each tool is to enhance both educational and research activities. This thesis chronicles the development of one such tool aimed specifically at the design and analysis of computer communication networks. Specifically, the tool's purpose is to automate the mathematics of computer network design in order to provide rapid analysis of network design decisions. The work drew principally from three fields of study. Two of these areas are computer networks and software engineering. A third perhaps more subtle and possibly more intriguing area is that of the human-computer interface, part of the larger discipline of what has come to be known as human factors engineering. A primary concern during the development of the project was that the software produced be useful, not a trivial statement given the amount of software available that is difficult to learn and use. It became evident during the development of this project that, as a general rule, there is an inverse relationship between ease of use and program complexity. Therefore, much of the time and effort devoted to this work was directed toward the transparency of the human-machine interface.

Organization

The remainder of this chapter will provide some background leading up to a statement of the problem. It will also describe the approach taken to solve the problem as well as the tools and materials used in the development of the solution. The following chapter presents a review of the background literature with emphasis on past research efforts in computer network analysis and software development methods.

Following the literature review will be presented a chapter on the development of the network theory and the analytical models that were implemented. The succeeding chapter will provide a detailed account of the software development process itself. The final two chapters will present results and discussion and finally some ideas on areas in need of further study or refinement.

Background

Students in the computer communication networks sequence of study at the Air Force Institute of Technology (AFIT) are responsible for becoming familiar with the analysis of many different types of computer networks. Many of the points presented in lecture are more easily understood if the student is able to simulate networks through the use of graphics-based computer aided design (CAD) tools, thus making the mathematics transparent. There are presently available a variety of tools commercially that perform generalized network simulation and analysis but are not specifically designed to enhance the study of computer networks. These applications often require the knowledge of difficult commands or even an entirely new computer language, thus causing the learning curve of the software to interfere with the mastery of the subject material. In other words, power and ease of use seem to have an inverse relationship. Also, these packages are expensive and probably out of reach of most personal budgets.

Given the above observations it becomes apparent that there exists a need for a software tool available in the public domain targeted at network topology to supplement the material presented in an introductory course in computer networks. The desired outcome as defined by members of the faculty at AFIT is rapid analysis of network design decisions. Furthermore, the system developed should fit into the somewhat ambiguous genre of "user-friendly" application software.

Problem Statement

Presently there is no requirement for laboratory work to supplement the lecture in the introductory course in computer networks. In order to develop an effective laboratory program, useful tools should be either acquired or developed. Presently available software tools alone do not adequately fulfill these needs and therefore development of software tailored more to the educational environment is in order. The objective of this research is the creation of a PC-based CAD tool that automates the design, analysis, and evaluation of computer network topologies presented in lecture.

Approach

The area that received the largest amount of attention was that of the user interface. A search of the background literature revealed some basic ideas about what makes a given application easy and interesting to use. Although the interface went through a series of refinements the ideas presented in the literature continued to be the basis for the design and implementation. Presently, there is not a standard interface for the software tools created at AFIT and consequently the visual manifestation of the application developed was largely a statement of the author's own ideas. Several ideas were implemented and tested iteratively until a final implementation was decided upon.

Another main concern involved the search and acquisition of the necessary development tools that supported graphics, mouse operation, and printing. Emphasis was placed on the level of support provided, guided by the notion that enhanced software development tools should encourage the development of better software.

After the software tools were selected but before the actual development of the software began, some time was spent iteratively writing and testing small routines that performed the low level tasks. For example, in order to draw a computer network on the display several graphics primitives were tested. It became apparent that some primitives required more computational power (and hence time) to create than others, particularly when floating point operations were involved. One outcome was the decision to use rectangular rather than circular objects to represent the nodes of a computer network.

Next it was necessary to research the computer network theory problem set to be modeled or simulated. This work relied heavily upon the textbooks currently used in the computer network sequence of study at AFIT. The outcome was a set of equations that were later mapped into the software solution.

Given the set of software tools and the set of analytical models, the actual development of the software could proceed. The process required that the requirements be organized and then mapped into a preliminary set of structure charts. From the structure charts a process of iteratively coding and testing the software was undertaken. Finally, the finished product was put through a series of formal tests as well as a field test involving the students in an introductory course in computer networks. Throughout the implementation phase, advice was solicited from fellow students as well as the thesis advisor and other faculty members.

Materials and Equipment

The Digital Engineering Laboratory at AFIT has a number of Z-248 microcomputers equipped with mice and Enhanced Graphics Displays (EGA). Although there are more powerful machines available in the lab, the Z-248's were more plentiful and used the popular MS-DOS operating system, already familiar to a majority of the students. Using these machines the software can reach a larger audience and the PC was therefore chosen to host the application developed in this study. The actual development was done on a 10 MHz 8088 based PC compatible machine equipped with a 30-Megabyte hard disk, EGA display, and Logitech mouse.

The software requires a PC compatible microcomputer with either EGA (256K) or Hercules graphics, and a Microsoft compatible mouse. To produce a hard copy of the drawings, the printer must support IBM graphics.

The software was written in Turbo C, version 1.5 customized with the CXL extended language library, version 4.0. According to its licensing agreement, the CXL package may be used for non-commercial software development without the payment of licensing fees. A copy of the package has been submitted with the simulator source code to the Department of Electrical and Computer Engineering in the School of Engineering at AFIT.

The Turbo C package is a commercial product and must be licensed before it can be used. The compiler has extensive graphics support while the CXL library provides mouse and windowing functions. The graphics driver (VGAEGA.BGI) provided with the C compiler was converted to an object file and added to the graphics library, thus eliminating the need for a separate device driver to execute the program.

II. Review of the Literature

The background literature for this effort is divided into two broad categories: computer network analysis and software design. The first subset is concerned with the object to be modeled, that is, the network itself. Because the product of this thesis will support a specific course sequence at the Air Force Institute of Technology, the thesis advisor and faculty members have clearly defined the type of networking problems to be studied. The second subset, that of program design and implementation, deals with software engineering issues and is well supported by past and present research efforts.

Computer Network Analysis

M/M/1 Queueing Systems. In order to simulate the arrival of traffic into a network a model must be devised that reasonably approximates a real world situation. One of the most widely used queueing models is the M/M/1 model attributed to A.A. Markov and described by Kleinrock and others. This model is popular due to its "memorylessness" property; that is, the system is completely defined by its present state and therefore the point of time at which study of the system begins is irrelevant. The fact that the analysis does not change as a function of time greatly reduces the amount of analytical complexity and permits mathematical models that are tractable. The continuous M/M/1 system is characterized by exponential distributions of both time between arrivals and service time (7:94-98). Tanenbaum found that an exponential interarrival time density is a reasonable approximation to data packet arrivals in a network and that the same distribution can be applied to service times as long as those times do not become exceedingly long (11:58).

Design. Tanenbaum presents an introduction to the backbone design of a data network in his text Computer Networks. The process begins by generating a starting topology based upon geography and then refining the topology with an optimizing heuristic such as the link deficit algorithm. The link deficit method assigns links between stations in such a way that the physical lengths of the links are minimized while also minimizing the number of links connected to each node. Flows are then assigned using the shortest path algorithm which requires that the data traffic use the shortest routes available. The final step is the calculation of the optimal capacity assignments for each link based upon the desired average delay for the system. Tanenbaum's method is based upon static routing paths; however, as long as the network is in a reasonable state of equilibrium the model should approximate a network that routes its traffic dynamically (11:61-71).

Software Development

The software development of this project draws from the areas of software engineering, database design, computer graphics, and the study of the human-computer interface. The goal of this literature review was to determine the present state-of-the-art in these areas as they apply to the computer environment chosen for this work (the PC-compatible microcomputer).

Software Engineering. The engineering of high-quality software can be broken down into four major subactivities: requirements analysis, design, implementation (coding), and testing. The primary purpose of requirements analysis is to decompose a complex problem into pieces that are easier to manage. There are a number of different ways to attack the problem, ranging from problem-statement languages to graphical CAD tools. The desired result is a set of specifications or diagrams that describe what the software is required to do (9:137-142).

The second subactivity, that of design, is concerned with the structure of the program, independent of coding concerns. The design stage is important not only to the development of the software but also is critical for the maintenance of large software systems. Two methods currently in use are object-oriented design and top-down design. Object-oriented design (OOD) creates a model of the real-world problem and then maps it into the solution space. It is a descendent of data-structure oriented design where appropriate data structures are determined before any function generation begins (2:47-50). OOD does not require that the designer map the real-world objects into pre-defined structures, rather, it allows the creation of abstract data types. Only after the operations on those objects are defined does the creation of implementation details begin (9:357-358).

Schildt recommends a top-down design approach based upon levels of functionality rather than levels of objects. Using this method the designer starts with a general description of the problem and then works downward, gradually adding specifics. The first module designed should be at the highest level of functionality (10:835-837).

The third area in software engineering is that of implementation. One very popular method is incremental testing, made possible by the proliferation of interactive terminals and personal computers. In incremental testing the developer builds the software little by little, testing as development progresses. This method has a distinct advantage over that of batch submission in that errors are caught while they are made and are therefore easier to track down, resulting in a reduction in the debugging time required (11:881). Booch advocates bottom-up development because it promotes the creation of shared elements or tools (2:390). When coding, Johnson recommends that the problems be first modeled as algorithms in order to avoid the distractions caused by attention to details required when writing correct code (6:121).

The final phase of software development with respect to engineering aspects is that of testing. Testing can be divided into two types - black box and white box. In black box testing the one conducting the test is not concerned with the inner workings of the software. Test cases are sets of input values that are expected to produce certain outputs. Normally, the test cases are determined by the creation of both valid and invalid equivalence classes. The concept can be taken one step further by using values that are on the boundary between valid and invalid input, (this method is sometimes referred to as boundary value testing)(9:485-486).

White box testing, on the other hand, is concerned with the operation of the internal structure of the programs. Test data is chosen such that all control structures within the code are considered. White box testing therefore requires that all paths through the program be tested. Obviously, this method presents logistical problems due to the combinatorial explosion of the number of paths through a large program (5:999).

Howden recommends a functional approach to program testing. He suggests that a program be thought of as a collection of interacting functions, where a function is defined as a set of operations that transforms data. This method requires that the programmer or whoever conducts the test have knowledge about three areas related to the program. First of all, as in black box testing, there must be knowledge of what outputs values are expected from a given set of inputs (input-output). Secondly, there must be knowledge of the proper sequence of function calls (trace). Finally, it must be known which pairs of events cause specific data transformations (interface) (5:997).

In functional testing all functions in the program must be identified and tested. Furthermore, all combinations of statements within the function must also be tested. The functional test data should include extreme and non-extreme cases tailored to find faults in the constructs that make up each function. The big advantage over traditional white-box testing is that not all paths in the program must be tested separately. Rather, each function is tested individually, thus avoiding the combinatorial explosion of test cases mentioned earlier (5:999).

Database Considerations. In Adams' thesis a study was made concerning the type and availability of a data base management system (DBMS) that was applicable to the public domain and educational environments. Because much of his work parallels this effort some lessons can be learned concerning which direction to take. Adams' study found that it was better to develop a simple original design rather than to implement an off-the-shelf package. One reason is that existing packages are more powerful than really needed and therefore consume an unnecessarily large amount of the memory resources of the computer. Another reason is that procurement of such a system would cause logistical problems due to copyrights and licensing requirements (1:2.2-2.4).

Implementation of Graphics. To guide the philosophy of the graphics design, the work by Johnson provides a useful framework to apply to the development of the graphics software for this system. Johnson divides a graphics oriented program into three "worlds". The first such world is what he calls the "world of control" which provides the means to alter the program flow. The editing and selection activities are examples of this environment. The second "world" mentioned is the "world of reals" or floating point representations of locations and attributes. The third and final realm is the "world of displays" where all numbers are of type signed integer, suggesting that there may exist a mapping from the world of reals to the world of displays but not vice versa (due to round-off errors). Locations and attributes are presented to the graphics hardware in the world of displays (6:40-41).

Human-Computer Interface. The Adams' study iterates some basic rules concerning the development of the human-computer interface. These rules all suggest that the usefulness of a software package is strongly dependent upon the user's perception of the ease of its use. The emphasis must be placed on keeping the user motivated. There are a number of features explained in the study that apply to this project. Such features are the use of the "mouse" as the preferred screen manipulation method, the inclusion of on-screen directions that supplement the user's manual, and such built-in features as confirmation of program termination before saving files. (1:3.12).

Carroll and Mazur conducted a study of first time users to a new computing environment. They found that novice computer users fear that improper actions may some-how damage the software or hardware. Also, most users new to a program or system will try to avoid reading the documentation. The new users typically tried to go it on their own only to later become frustrated and begin skipping around in the manuals hoping to find the answers to their problems.

The study produced some recommendations concerning the writing of programs. First of all, the writer should pay attention to the level of expertise of the user. The instructions should be short and not too regimented. Also, it is wise to avoid the inclusion of positive or negative reinforcement messages within software. If taken out of context or repeated, the messages can become annoying. Error shielding is a term used to describe the amount of built-in protection in a program. "Overshielding" occurs when the software attempts to anticipate the user's improper actions and then invisibly makes corrections. The user becomes confused because the outcome of an action did not correspond to the (erroneous) input. (The machine made the change without notifying the user). The opposite end of the spectrum is "undershielding", where the user is allowed to go ahead and make mistakes with no action being taken. The programmer should also avoid error messages that are too general. The error messages should be comprehensive and all at the same level. Finally, the use of "cute" phrases should be avoided because the novelty of such messages soon wears off.

As far as terms and language in general go, one should avoid arcane labels. For example, the substitution of "fixed disk" for "hard disk" may be more accurate but it is not the commonly accepted term. Labels and commands should be simple and to the point. Terms like "save", "close", and "print" are not easily confused and they adequately describe the actions to be taken. In regard to the use of a mouse, the programmer should make sure that the mouse sensitive areas of the screen are large enough to be easily selected. One should assume that the user is not experienced in working with a mouse. Finally, good software should include provisions to undo actions taken in error (19:36-46).

Summary. The purpose of this chapter was to relate this thesis effort to previous research. The preceding paragraphs attempt to summarize the body of literature investigated and gives some insight into the approach taken in this work. The intent is not to follow a predefined path but to choose a mix of the best methods.

III. Network Theory

This chapter will present the analytical model that was mapped into the software solution. It is important to note that a majority of the effort was put into the design and implementation of the user interface, while implementation of the networking algorithms was relatively straightforward. Consequently, other methods of analysis may be implemented without much difficulty using the interface developed in this work. This subject will be discussed further in the sections on design and recommendations.

This chapter does not attempt to provide the detailed development of the queueing and network theory behind the equations used. The reader is referred to Tanenbaum (2) and Kleinrock (7) for more depth if needed. The focus of this chapter is on what assumptions have been made and how they affect the accuracy of the analysis. It also presents an explanation of the various equations selected for implementation into the software solution.

Networks of M/M/1 Queues

Network Model.

Before discussing the theory that drives the problem-solving algorithms of the software, a discussion of the specific type of networking problem is in order. The network to be modeled consists of two basic elements: switching processors and transmission lines (2:7). These elements are manipulated as nodes and links, respectively. Each node receives traffic from two sources: the incoming links connected to it and/or the host computers that access the network through it. Messages are made up of a series of packets of varying length. Figure 1 shows the translation from the physical network to the abstract model for analysis.

The routing algorithm used is static. From the standpoint of an originating node, all messages intended for a specific destination will use one or more predetermined sets of connecting links. No routing decisions are made at intermediate nodes along the route

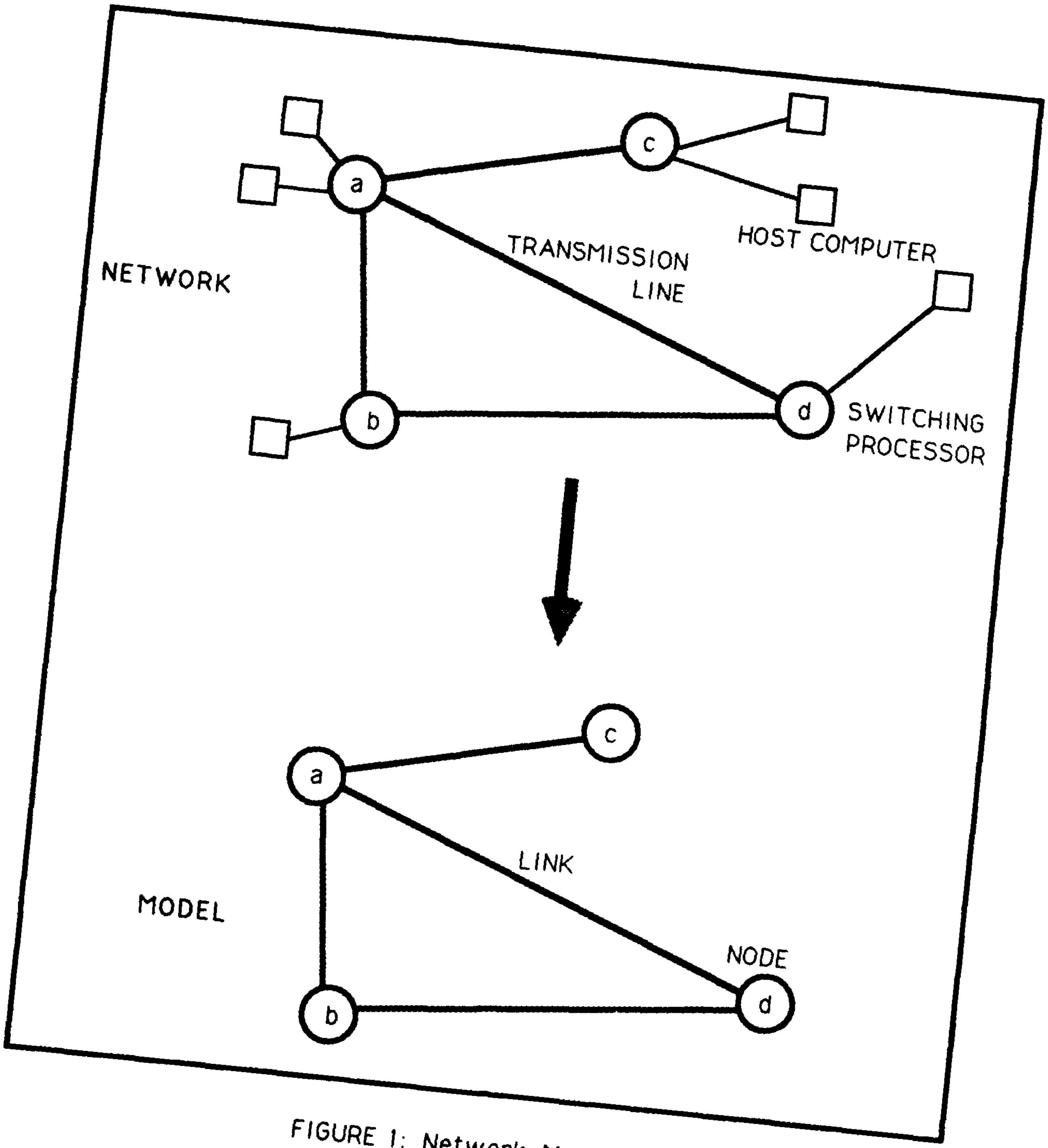


FIGURE 1: Network Modeling

taken. A node accepts all packets as they arrive into its infinite queue. If the packet is destined for some other node it will be forwarded along the link specified by the packet's routing as soon as the link is free. This type of network is sometimes referred to as a store-and-forward network (2:8).

In order to develop a solvable mathematical model, several simplifying assumptions are necessary. First of all, the number of arrivals during any interval of time is an independent random variable (4:50). The steady-state arrival of packets therefore constitutes a Poisson process and is described by exponential interarrival intervals. This assumption appears to be fairly accurate for large numbers of independent customers. As far as the processing of the packets by the node, the exponential service time assumption also holds as long as service times are not too long. Stated another way, the distribution of bits per packet is also a Poisson process (2:58). Therefore, given exponential interarrival intervals and service times and the added condition that each link can process only one packet at a time, the M/M/1 queue serves as an adequate model. Propagation delay, check sum calculations, errors, retransmissions, and control overhead are all disregarded.

The next problem is how to treat the interconnection of the multiple Markovian servers that make up the communications network. Given that message service (or packet length) is a Poisson process each link can be considered as just another stream of arrivals to the node - a parallel combination of arrivals to an M/M/1 queue. Burke discovered that multiple Poisson arrivals can be added together and then treated as a single large Poisson process with its mean arrival rate equal to the sum of the rates of the arrival streams feeding it. The weakness with applying this idea to the model is that there is an assumption that packets are discarded and then regenerated at each node, with new packet lengths being assigned according to an exponential distribution. This approximation is known as the Independence Assumption and has been shown by Kleinrock to be reasonably accurate (2:63). In reality each packet usually remains constant in length from its point of origin to its destination. There are exceptions. For example, gateway processors connecting networks may modify the size of the packets in order to accommodate the different standards of the interconnected networks.

Network Parameters. Having described the type of network to be modeled, the next step is to determine the parameters of interest and develop the equations necessary to provide the analysis. The input variables are shown in Figure 2. From the standpoint of a

Input variables:

γ_i	: traffic for route i (packets/sec)
$\frac{1}{\mu}$	: average packet size (bits/pkt) for network
C_i	: link capacity in kbps (kilobits per second)
μC	: service rate (packets/sec)
λ_i	: average arrival rate (packets/sec) for link i (Sum of all traffic γ using the link)
d	: cost per kbps (Normalized to 1).

FIGURE 2 : Network Parameters (Input)

user on the network, one parameter of obvious interest is how long does it take to route a message through the network. The answer requires a calculation of the mean queueing delay. To calculate delay the average arrival rate of the packets must be known as well as the average packet length. Also the traffic between each pair of nodes is needed since each route places additional load on the links that it uses. The network must be in a steady-state, that is, the number of packet arrivals to each link must be less than the average rate of service at the link. The number of arrivals includes all traffic that uses the link as determined by the routing table. The ratio of packet arrivals to rate of service is another parameter of interest known as traffic intensity and gives an indication of the amount of operating headroom of a particular link. These parameters are summarized in Figure 3. By determining the amount of delay and traffic intensity at each link, the behavior of the network can be visualized. The results of the calculations can be tabulated and then comparisons can be made.

There is an important parameter of interest that relates to the operation of the network as a whole, and that measurement is the average delay for a packet entering the network. Although this measurement is only an approximation it is nevertheless a way to monitor general system performance. The next section will discuss how the average packet delay for the network can serve as a constraint for controlling network costs through flow and capacity assignments.

Figure 3 shows the formulas for average link delay, traffic intensity for a link, and the mean packet delay for the network. The reader is referred to Tanenbaum for a detailed development (2:62-65).

Flow and Capacity Assignment

Flow assignment is largely determined by some type of heuristic process and will not be mentioned in detail here. What is important is that some type of routing algorithm (only static is considered) is needed to assign traffic in the first iteration of the network design. One example of such a heuristic is the shortest path algorithm. In this method routing between any two nodes is determined by a combination of shortest geographical distance and the amount of traffic already assigned to the links on the net. After all routing has been accounted for, the design has a beginning topology to which the analysis can be applied.

Calculated parameters:

$$T_i = \frac{1}{\mu C_i - \lambda_i}$$: average delay for link i

$$\gamma = \sum_{i=1}^{m} \gamma_i$$: sum of all traffic on the net

$$\lambda = \sum_{i=1}^{m} \lambda_i$$: sum of all link arrival rates

$$\rho_i = \frac{\lambda_i}{\mu C_i}$$: traffic intensity for link i

$$T = \frac{\lambda}{\gamma}$$: mean packet delay for network

$$C_i = \frac{\lambda_i}{\mu}\left(1 + \frac{1}{\gamma T}\,\frac{\sum_{j=1}^{m}\sqrt{\lambda_j d_j}}{\sqrt{\lambda_i d_i}}\right)$$: optimal capacity for link i based on delay constraint T

FIGURE 3: Network Parameters (Calculated)

Using an iterative process of routing assignment and network analysis the design can be optimized. After a proposed topology has been determined by whatever means, the next step is to calculate the parameters mentioned in the preceding paragraphs. The results will probably show that some links on the network are more heavily used than others which suggests that there is inefficiency in the design. A heavily loaded link will introduce a large delay component while a lightly loaded link may be wasting resources. The last equation listed in Figure 3 provides a means to more efficiently assign capacities to the links, subject to an average packet delay that is acceptable. The equation assumes that there is a linear relation between cost and kbps capacity. The results are assuming continuous capacities and should be adapted to the reality of discrete capacity assignments. The reader is referred to Kleinrock for the detailed development of the minimization technique used (2:73).

Summary

The material presented in this chapter as an overview rather than to attempt the provide all of the detail necessary to develop the network queueing models. The goal was to present the set of equations that were translated into the number crunching portion of the software. In Figure 3 these equations are summarized and will be referred to in the software development discussion which follows.

IV. Software Development

Requirements

Requirements were determined largely by the thesis advisor and other members of the faculty who regularly instruct students in the computer network sequence. It is important to note that not all of the requirements were determined before the development began. During the course of the work capabilities were discovered and new ideas were explored on almost a daily basis. Because a conscious effort was made to adhere to software engineering principles, the changes in requirements were implemented without major disruptions.

General Requirements. One of the main requirements was that the software be compatible with computers in the Digital Engineering Laboratory. The Z-248 microcomputer with Enhanced Graphics Adapter (EGA) display and Microsoft compatible mouse was the targeted environment. These machines fall into the general category of IBM compatibility which has a large installed user base. Because the software developed is part of the public domain, it can be copied freely and run on any machine having the same capabilities.

Program flow was determined to be as follows: the user first must create a visual interpretation of the network to be monitored. In most simulation applications, this actvity requires a two-step approach where the user must create an abstract model independent of the software and then translate this visual model into a set of statements that describe the system. In this project, however, the requirement was that the development of both the visual representation and the description of the network be a one-step process.

The drawing produced must show both the switching processors on the net as well as the interconnecting links. Each of the links must be assigned a data rate capacity in kilobits per second (kbps). After the network is created and edited to match the problem, there must be a means to provide the input of messages into the network and subsequently route the messages to predetermined destinations. The routing is static, that is, all routing for packets arriving at each node is determined before it enters the net. The alternative is dynamic routing where each node must make decisions as to which link to route each packet as that packet is received. In order to provide additional flexibility in the analysis, the traffic arriving at each node must be in packets per second with the packet length being a variable quantity in units of bits per packet. The end result of the graphics portion of the software must be a visual representation of the network with assigned link capacities and a routing table indicating traffic throughout the net.

Concerning the network analyzer segment of the software there are several parameters developed in the network theory portion of this work that provide performance measures and are the objectives of the simulation. The parameters desired for each link are average delay, traffic, traffic intensity (ratio of traffic to capacity), and optimal capacity assignment based on an average delay constraint. In addition, average packet delay for the entire network is needed. These parameters were discussed at some length in the preceding chapter.

User Interface. The human-computer interface is difficult to evaluate and almost impossible to quantify. Nevertheless, there were some general requirements concerning the link between the user and the analytical engine of the software. Most importantly, a short learning curve and ease of use were the dominating factors in the design of the user interface. More specifically, the network components must be displayed rather than described by some obscure simulation language.

The use of a mouse to operate the application was determined from the onset to be more desirable than the keyboard for creating graphic images; however, some actions such as data and text input are only implemented in a reasonable manner by using the keyboard. The decision was made that the mouse be the primary input device with use of the keyboard restricted to input of text and numerical data. As a general rule, the keyboard was to be used only when the mouse could not perform the same function.

The process of inputting commands should be as passive a process as possible. Menus and context sensitive prompts should be used whenever needed to let the user focus his concentration on the purpose of the software rather its operation. Readily available on-screen help displays with concise command explanations also help satisfy the user-friendly interface requirement.

The final "look and feel" of the application was not defined as a requirement but rather evolved during the design and implementation stages. The goal was to make an application that was visually interesting to use. More about this will be mentioned in the discussion on the implementation details.

Other General Requirements. There were other requirements concerning the operation of the application. It is desirable to have the capability to save the drawings, routing tables, and analytical results to disk for later retrieval. The database itself must be of a dynamic nature, that is, the memory and disk requirements should be determined by the size of the data (static arrays are to be avoided). This requirement allows network files to be limited only by the amount of memory available.

Finally, the design of the software must be easily upgradable; it must be constructed such that future improvements are facilitated by the structure of the program. For that reason, particular attention was given to adherence to accepted software engineering practices. This requirement infers that sometimes trade-offs are made between speed and modularity. Function or procedure calls reduce execution speed due to the overhead of transferring control to another segment of code (saving registers, status, program counter addresses, etc.) but make the source code more readable than long stretches of (faster) in-line code. A general rule was adopted that if a single function exceeded one page of code, it was probably too long and should be divided into smaller functions.

It was expected that throughout the development of the software, changes would be requested by prospective users. Modularity of the design as well as other well accepted software engineering practices greatly enhanced the modifiability of the code. During the coding phase several suggestions were indeed made by faculty members and colleagues in the lab. These suggestions were, for the most part, implemented without greatly affecting the rest of the code.

Summary. The set of requirements as outlined defines what the application is to accomplish. The overriding objective was to develop a tool that could be quickly learned, was interesting to use, and performed the job for which it was intended.

Design

The design process was divided into three major categories. They were 1) the creation of the objects and their operations, 2) how the data was structured and 3) program structure.

Operations on the Network Objects. Before developing the database it was first necessary to determine the objects and their operations. In a vector graphics design approach such as the one used in this project, images are not stored as collections of pixels (picture elements). Rather, each object is given a set of attributes that is accessed each time the object is redrawn. Each collection of attributes can be thought of as a set of directions for a set of tools. For example, in this work one of the objects is the network node, and each node must have a set of attributes that uniquely describes it. Each node is defined simply by its position on the x-y plane of the screen grid along with a single character identifier. Given the above information the node can be reconstructed using some type of routine that can draw and fill a graphic primitive and then label it. All else that is needed is a reference or pointer to its position in memory so that the program can access it.

There were three major objects created. As just mentioned, one of the objects was determined to be the network node, which corresponded to the message switcher of the physical network. The other objects are the data link (transmission channel) and the route (static routing description). The link is the basis for the analytically intensive operations of the application and is described graphically by its starting and ending points on the screen grid. Its other attributes are the parameters as mentioned in the chapter on network theory - link capacity, traffic load, traffic intensity, average delay, and optimal capacity assignment. The calculations of these parameters are additional operations on the link object. The third major object is the route, which is composed of an ordered set of nodes and a measure of the traffic routed between the endpoints.

When discussing the vectored approach to the graphics design the question arises as to what type of drawing tools are needed. The software development therefore temporarily departed from the usual sequence of completing the design before considering implementation details and focused on developing a set of tools to perform the low-level graphics. In order to establish the graphics potential of the application some time was spent in the early stages researching methods of producing primitives (circles, lines, rectangles, etc.). AFIT has developed in-house a set of routines that perform these tasks. These routines were written to be compiled with either Microsoft C or C86. The other environment considered was the Turbo C compiler, Version 1.5, which at the time had only recently been released. The Turbo C environment offered a rich set of graphics routines that also provided all of the functions needed. Due to the author's familiarty with the package along with other factors the Turbo C system was selected, although either environment could have been used. There was also no specific requirement that the software be written in the C language, but one of the personal objectives of the project was for the researcher to become more proficient in C programming. There are other development environments for the PC microcomputer that provide at least the same level of graphics support.

Database. A linked list was selected to represent the objects in memory. This data structure supports the creation of arrays of variable sizes. Memory is allocated as each element is added to the database and freed each time an element is deleted (dynamic allocation). It also allows data to be inserted in order and deleted without manipulation of the entire file. As elements of the list are deleted or created only a few pointer updates are necessary. A linked list may be singly linked, with a pointer to the next element, or doubly linked, with pointers to both the previous and the next elements of the array. The double-linked list was chosen because it can be searched in either direction, and it has the added security of being constructed either forward or backward using separate sets of pointers. It turned out that a single linked list would have worked equally as well and would have also had the advantage of consuming a slightly smaller amount of memory (2 bytes per element due to one less pointer per element).

Figure 4 depicts the structure of the data base used for the network. There are three data elements corresponding to the sets of nodes, links, and routes. The node and route structures were relatively simple, having few elements. Note that the route structure has no graphical information since all of the graphics details of the network are defined in the links and nodes. The link element contains most of the network information and is largely a result of the mapping of the parameters presented in the chapter on network theory.

Toplevel Design. Figure 5 depicts the original highlevel design. The highest level units are an executive module ("Topmenu") and three subordinate units ("Drawnet", "Defroute", and "Analyze"). The executive module handles filename selection and provides a menu to give the user the choice of which of the three lower modules to run. The solid lines show the lines of control as well as the hierarchy of the design while the dotted lines show that control can be passed freely among the three main modules while manipulating the same set of files. The modules shown subordinate to the three main

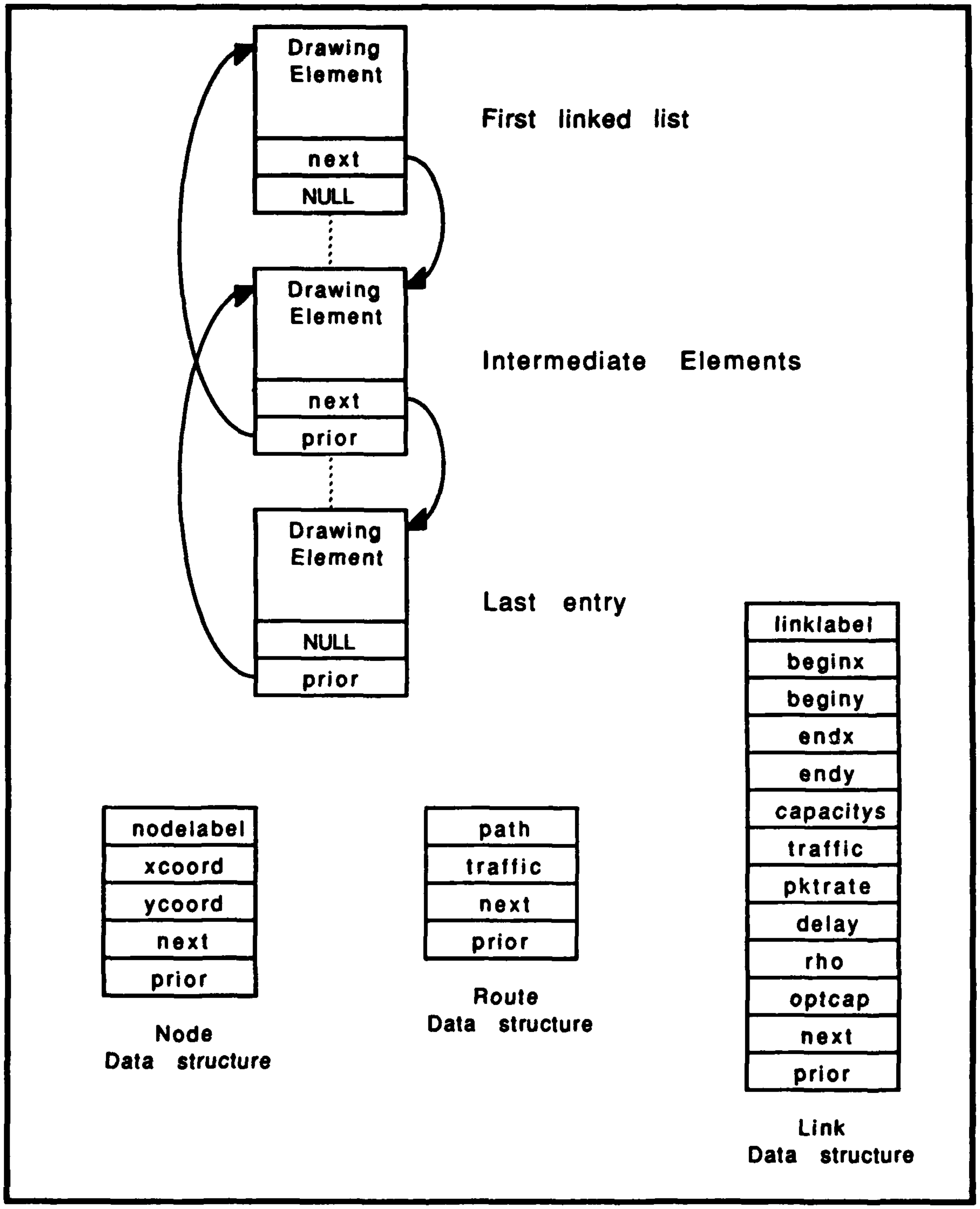


Figure 4. Data Structure Design

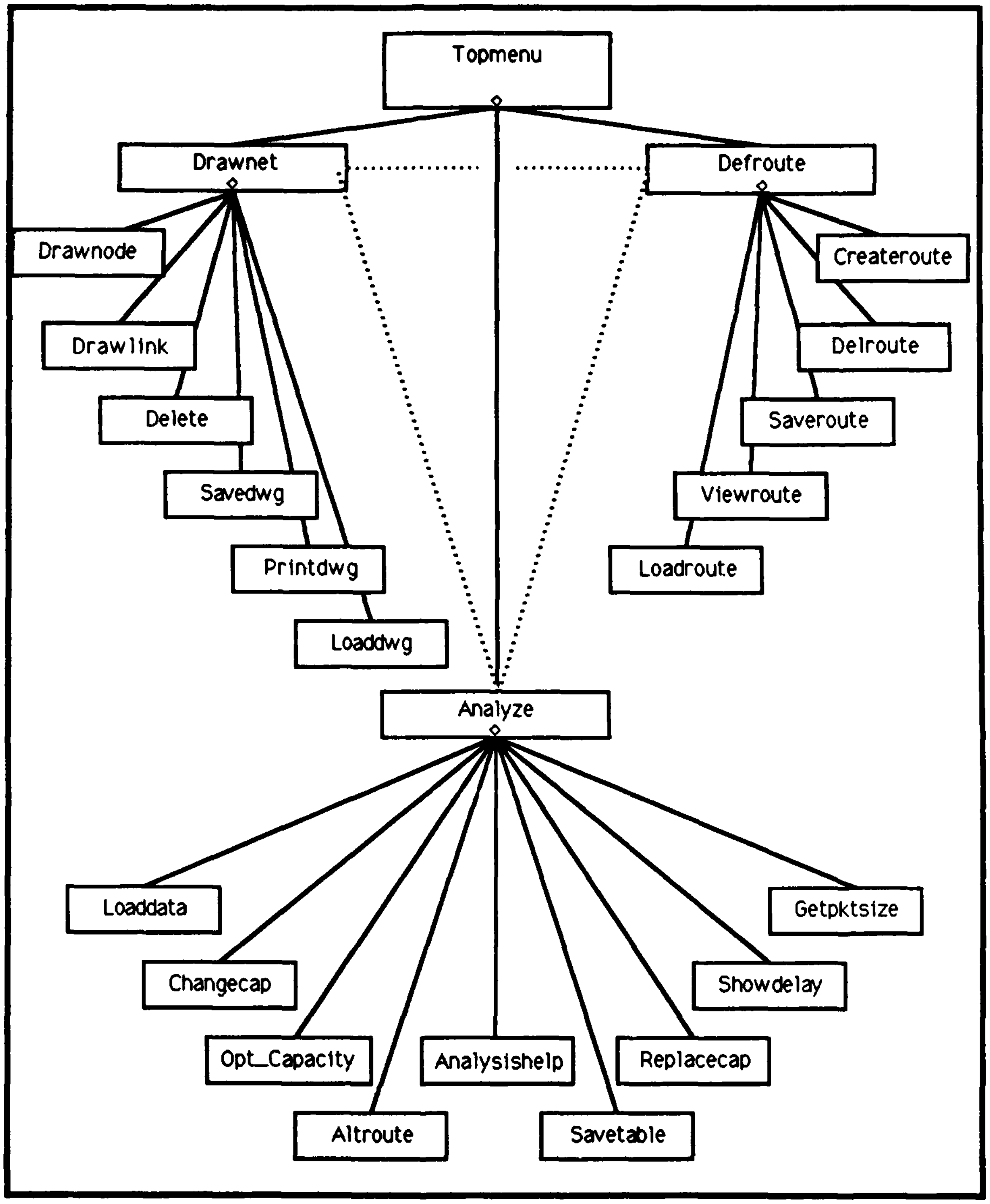


Figure 5. Structure Chart - High level design

modules were created to handle tasks that were determined to be necessary early in the design. More activities were created during the implementation phase.

The three main modules divide the operation of the application into three functional units. The first module, "Drawnet", is concerned with actually creating and editing the network to be studied. Like the top level executive module it too is a selection and decision point. Subordinate to the "Drawnet" executive are several modules that perform the major operations on both the graphical elements and the files. The module names were chosen so as to be self descriptive. At the same logical level as "Drawnet" is the "Defroute" module and its purpose is to create and edit the routing matrix. Together "Drawnet" and "Defroute" perform all of the graphics-oriented tasks. The third module, "Analyze", is the number crunching part of the application. It applies the network theory formulas to the graphically created database.

Lower Level Functions. Subordinate to the module "Drawnet" (Figure 5) are defined six activities that divide up the process of creating and editing a network drawing. First of all, the disk files that contain the data must be either created or opened if already available. The "Loaddwg" module performs both functions by accepting a filename and then loading the file from disk or, if not already existing, by creating a new file. The file is transferred to memory by successively allocating storage for each data structure and then transferring the data from disk. The process continues until the EOF marker is read. It provides two types of error handling. The first "error" is when the file is not found in which case it simply creates the file as mentioned earlier. The second error condition is when there is insufficient memory available to load the data, in which case an out-of-memory error is displayed and control is returned to the calling module.

The "Savedwg" module reverses the "Loaddwg" process by transferring the data from memory to disk. In order to transfer the data it must first open the disk file. If for some reason the disk file cannot be found, an error is displayed. All data in memory remains available to the application after the save.

The remaining modules manipulate the elements of the drawing. First of all, the "Drawnode" module when selected reads the mouse position and waits for a button click. When the button is pressed, control transfers to a routine that draws the node (in whatever form later decided upon by the implementation) and inserts the node data structure into the database in alphanumeric order. The module must also prompt the user for a single character label and then correlate the label with the node's data structure. The reason for the single character node label will become clear when the routing module is discussed. When creating the structure of the node, pointers to the previous as well as the next element in the database are added.

The other major drawing element is the link and is also created using mouse input. The link module connects existing nodes and prompts the user for the capacity of the transmission line it represents. In its original form the module formed only full duplex links, but this task was modified later on during the implementation to include simplex links. At this point each link consists of only the endpoints, the position on the screen, and its capacity in kbps along with pointers to the previous and next elements in the link array. After the link is created, its structure is inserted in alphnumeric order in the link data array.

Any useful drawing editor must provide the capability to erase objects. This design provides a "Delete" module which serves as the executive for the modules that erase the nodes and links. More will be mentioned about the erase functions in the section on implementation.

The final submodule in the design of the drawing editor is the "Printdwg" module and provides the ability to make printed copies of a network drawing for later reference.

The second major module discussed is "Defroute", or the routing editor. It controls five submodules (Figure 5). First of all, the route data structure is similar in construction to the link and node data structures and the same file operation routines can therefore be adapted to routes. These activities are covered by the "Loadroute" and "Saveroute" submodules. Also, as in the drawing editor, there must be a method of creating the routing table and if the requirement of using the mouse as the primary input device is followed, a route should be created by pointing and clicking on each node along the desired route. This ability is provided for in the "Createroute" module. It also creates a character string made up of the labels of the nodes that it contains. This string is represented in the data structure by the "path" element (Figure 4). The module must also allow entry of route traffic, either in the forward direction or in both directions. One distinction between "Defroute" and the drawing editor is that here one is working with two things, the network drawing and the routing table, rather than the network alone. Consequently there must be a means of viewing the routing table either by displaying both the network and routing simultaneously or, better yet, by providing a second display that is called up smoothly and quickly in order not to disrupt the route creation process. The "Viewroute" module is reserved for this activity. Finally, as with the drawing editor, there is a module that deletes entries from the database ("Delroute").

There is one primary module remaining to be discussed - the "Analyze" module where, as mentioned earlier, the mathematical equations are implemented. "Analyze" accomplishes this by iteratively calculating the network parameters whenever variables are changed. In Figure 5 the reader will see that this module controls several subordinate functions. As in the other primary modules, there is a unit to transfer the data from disk to memory as well as another module to save the data to disk ("Loaddata" and "Savetable"). The remaining units have no analog in the other two primary modules and each unit will be discussed individually.

Before the analysis of the network can take place there must be a provision for entering the average size of the packets in the net. "Getpktsize" provides this ability as well as the option to modify the packet size later if desired. This module is automatically called when the Analyzer is invoked and no packet size has yet been entered.

Because there are several submodules with separate sets of instructions for the user to control it seems that some type of on-screen help is appropriate here. The "Analysishelp" module is intended to provide an on-call list of directions that correspond to all of the activities in the "Analysis" module. By using an on-call help function, the commands displayed on the main module are kept to a minimum of characters, thus freeing up the screen for the analysis results.

Again referring to Figure 5, the "Changecap" module allows the user to modify the capacity assignment of any link in the database. Using the formula given in Figure 3 for optimal capacity assignment, the "Opt_Capacity" module calculates the optimal capacity for all of the links upon input of an average network delay constraint. The calculated optimal capacities are then entered into the link database. Carrying this line of calculation one step further, the "Replacecap" module provides the user with the capability of replacing all of the capacity assignments for the links with the optimal capacity assignments as calculated in the "Opt_Capacity" module. The average network delay can be viewed at any time by invoking the "Showdelay" activity.

The one remaining module yet to be discussed is "Altroute" and will be treated differently. The intention of this module was to provide a path to an expert system that looks at the present state of the network and by using some type of intelligent algorithm, makes suggestions for alternate routing. This module was given a separate compilation unit during the coding of the application.

Implementation

Environment. As mentioned earlier in this chapter, the Turbo C development system was selected to implement the design. For the purposes of this project the Turbo C version 1.5 compiler provided most of the capabilities needed with the major exception of the mouse drivers. The Z-248's in the Digital Engineering Laboratory are equipped with Logitech Mice which are Microsoft Mouse compatible. In order to satisfy the requirement that the application be mouse driven a search was conducted in order to find the driver support needed. AFIT has developed in-house a set of mouse drivers written in C and assembly code. These drivers were originally written for the Mouse Systems Mouse and were therefore not selected.

Fortunately, there are several libraries of C functions available from bulletin board systems (BBS) and the national on-line computer services. Several libraries were evaluated before deciding upon the CXL Extended Library version 4.0. The CXL library seems to fill in many of the gaps in the Turbo C library. Most notably, it provides extensive mouse and windowing support. Turbo C 1.5 supports windowing but at a much lower level and is therefore more difficult to implement than the CXL functions. The CXL licensing agreement allows use of the code without paying a fee as long as the work is for non-commercial purposes. A copy of the library has been submitted along with the source code for the simulator.

The environment used for this thesis effort was a somewhat customized version of the off-the-shelf package. First of all, the BGI graphics driver file "EGAVGA.BGI" is usually required at run-time for all EGA or VGA graphics routines written in Turbo C. For this project, the driver was integrated directly into the graphics library of the C compiler by first converting it to a ".obj" file using the special application BGIOBJ.EXE that comes standard with the Turbo C package. Then the TLIB.EXE application was used to add the new object file to the Turbo C graphics library file. If this conversion is attempted, the programmer must make provisions for having the device driver integrated directly into the graphics library by modifying the source code that initializes the graphics system. The purpose for integrating the graphics driver was to simply eliminate the extra file needed at run time.

Both the graphics and CXL library files were integrated into the main library file (CS.LIB) of the Turbo C package. The CS.LIB library file contains most of the general C routines and supports the "small" memory model. The small memory model, which limits the code segment and data segment of the code to 64K each, seemed to work well. It supposedly produces faster executable code by making all jumps intrasegment.

Some mention should be made of display modes on the PC. The EGA card operates in two modes - text mode and graphics mode. In text mode all screen writes use the built-in character generating routines of the ROM BIOS. Resolution is 80 characters wide by 25 characters high. The toplevel and analyze modules were written using this mode. The graphics mode is a direct pixel manipulation of the screen and allows more flexibility and resolution. Resolution is 640 pixels wide by 350 pixels high. Modes can be mixed to a limited extent. The drawing and routing editors both work predominantly in the graphics mode with a top line of text mode characters added for instructional purposes. On machines that do not support separate floating point operations (using the 80x87 chip) the drawing of arcs is a computationally expensive operation. Thus the decision to avoid drawing circles and arcs was made early on. An exception was made when the use of arrows to indicate traffic direction was implemented. Each arrow was composed of either one or two filled arcs, depending on its orientation.

Approximately 2000+ lines of code were developed in this work and with the additional 3000+ lines of header code the total came to over 5000. Advantage was taken of C's ability to support separate compilation units by dividing the source code into three separate files. One file supported all of the graphics and implemented the "Drawnet" and "Defroute" modules (See Figure 5). The second file implemented the "Analyze" module while the smaller "Altroute" module was separately compiled in order to facilitate future development.

Methods. In the coding process a technique of iterative coding and testing was used. Beginning with the low level graphics functions, dummy executive programs were written to control logical grouping of functions. Small sections of code corresponding to the modules in the structure charts as shown in Figures 6 through 9 were written and tested for functionality. These structure charts were developed from the original chart shown in Figure 5, and resulted from the finalization of the design during the implementation process, thus providing a de facto test of the maintainability of the software while still under development. Because of the experimental nature of this project, following a rigorous design process in great detail before beginning the implementation was not possible. In the author's opinion, if development had strictly adhered to the classical sequence of requirement definition before design and design before implementation, the resulting software would have been more limited in its capabilities.

Due to the hardware intensive nature of this application, many of the functions do not fall within the proposed ANSI standard for the C programming language. The code did, however, make use of the ANSI recommended practice of function prototyping in order to enforce stronger type checking than that provided in Kernigan and Ritchie's original implementation of the language. Function prototyping checks all parameters that are passed to functions. All of the passed parameters must agree with the original function declaration prototype.

Global variables were implemented but kept to a minimum. Only those variables that were required access by a very large portion of the code were declared as global. The database and filename as well as the mouse status are examples. The reader is referred to

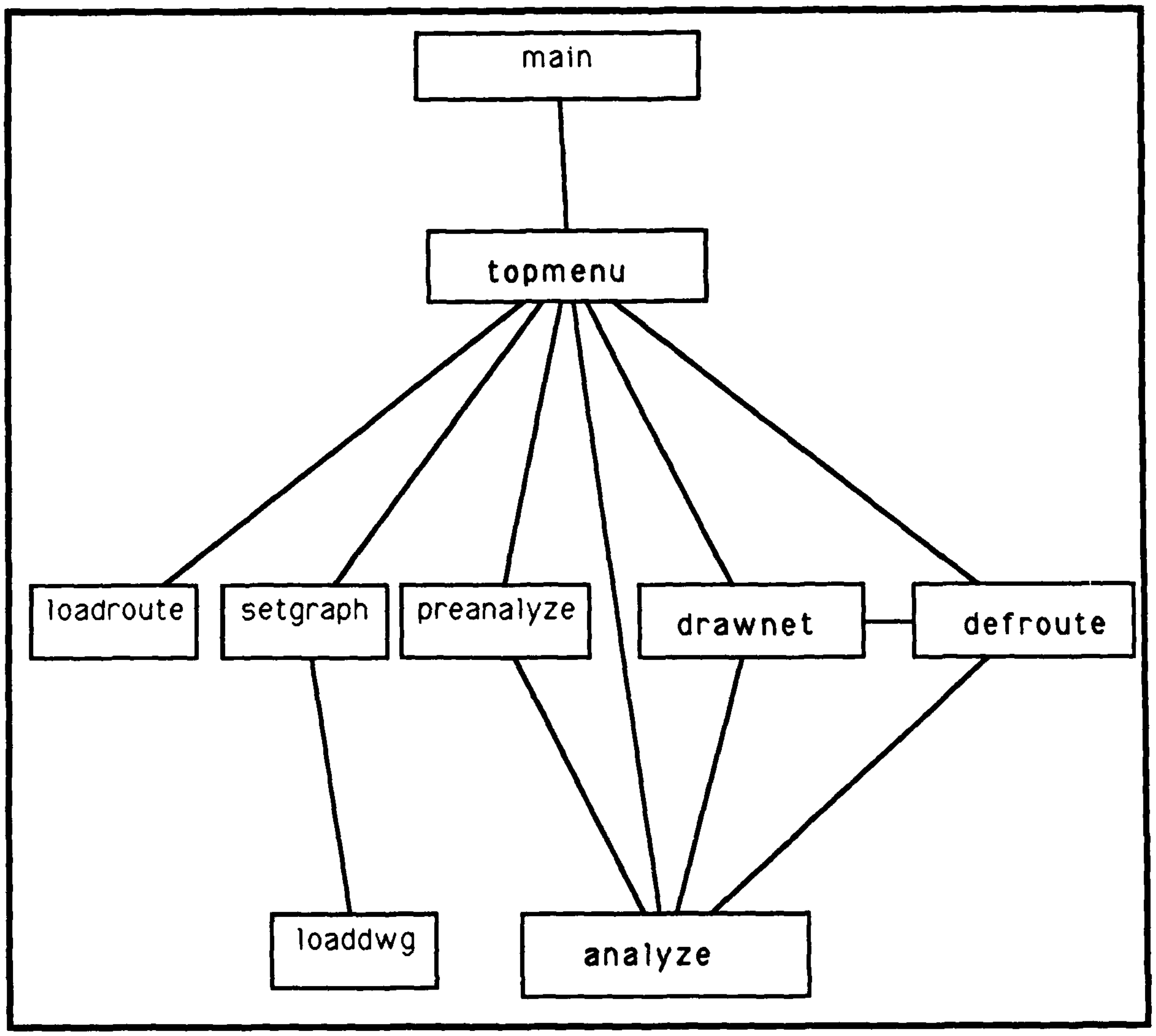


Figure 6. Structure chart - High level

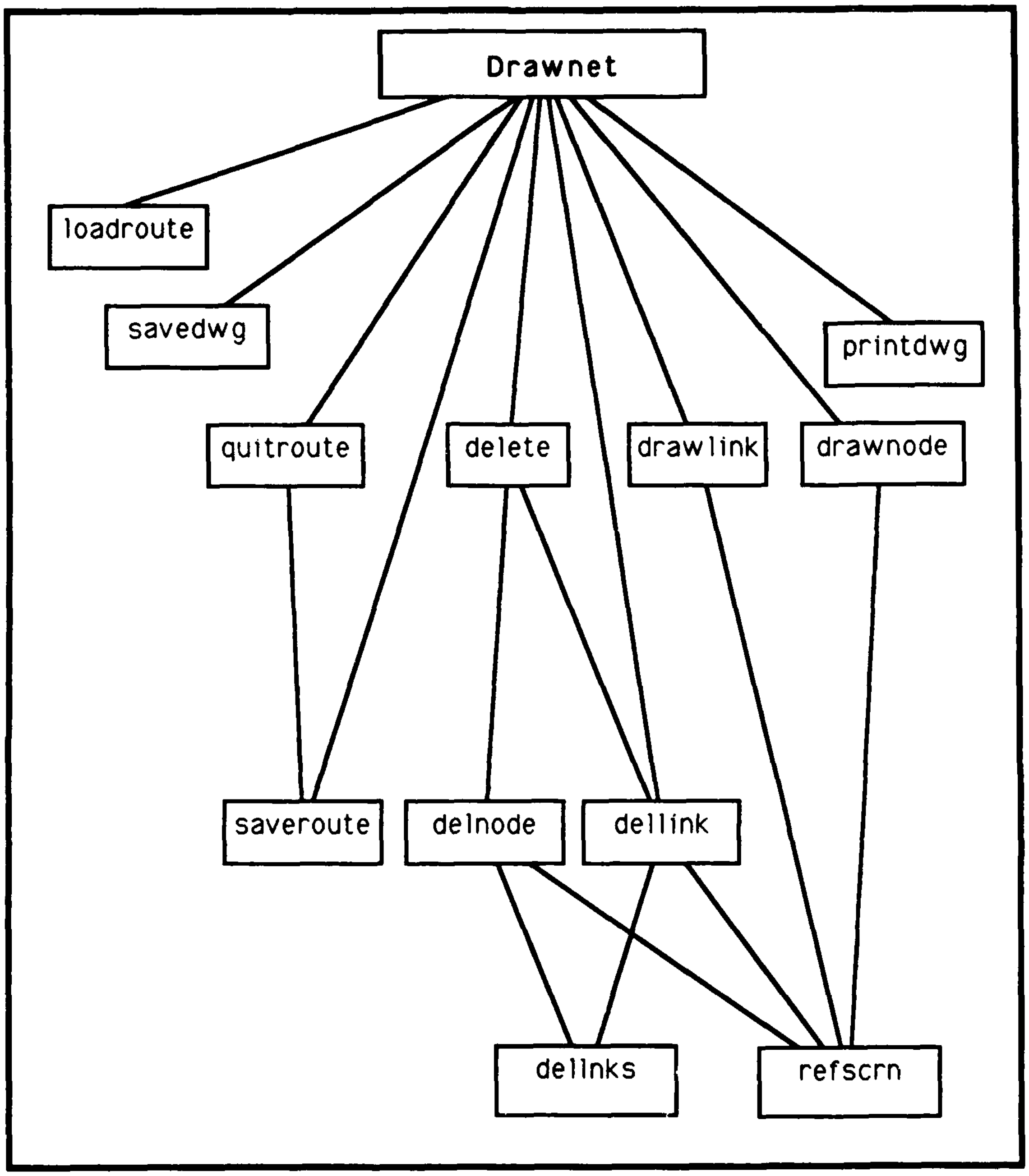


Figure 7. Structure chart - Network editor

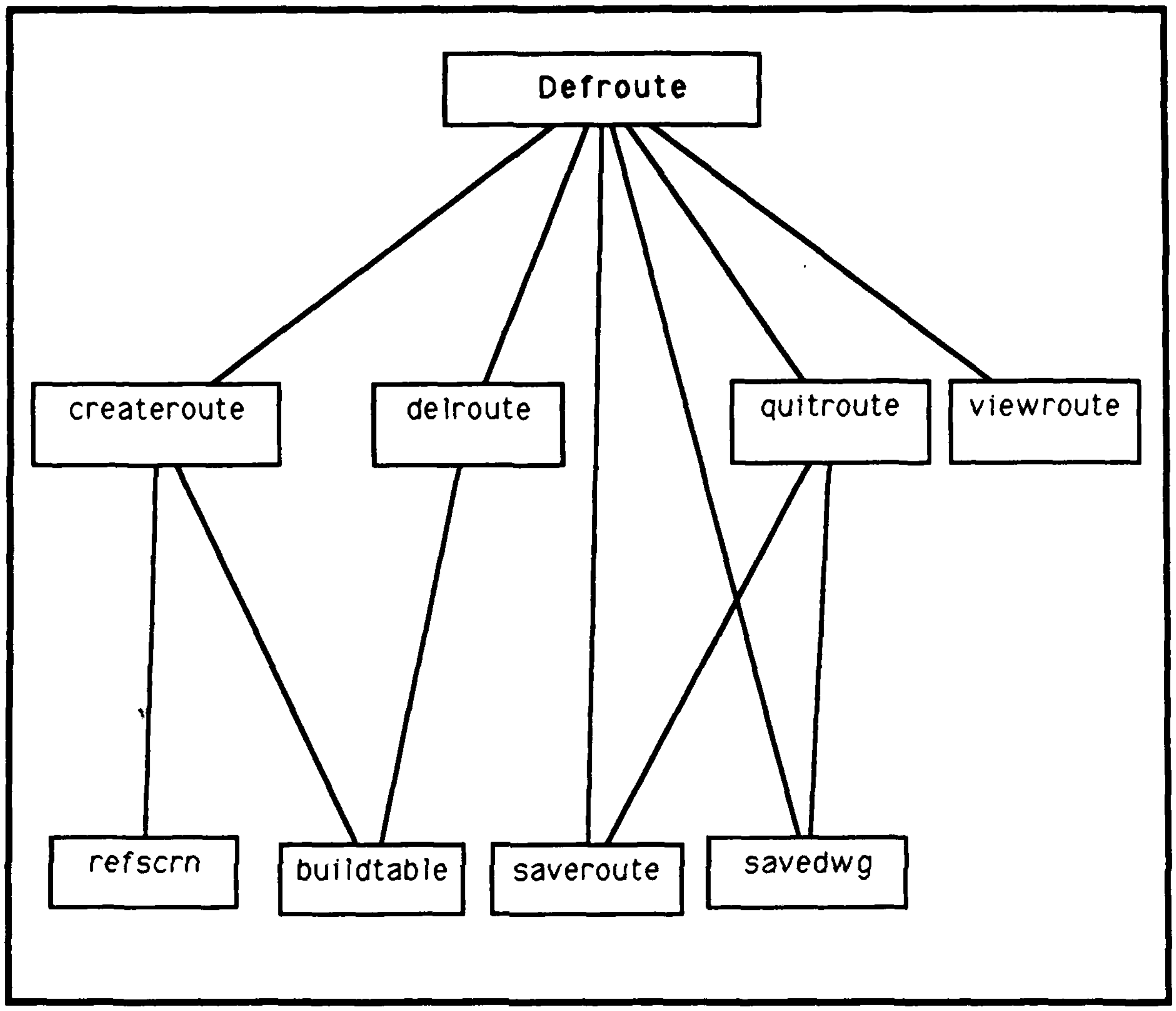


Figure 8. Structure chart - Routing editor module

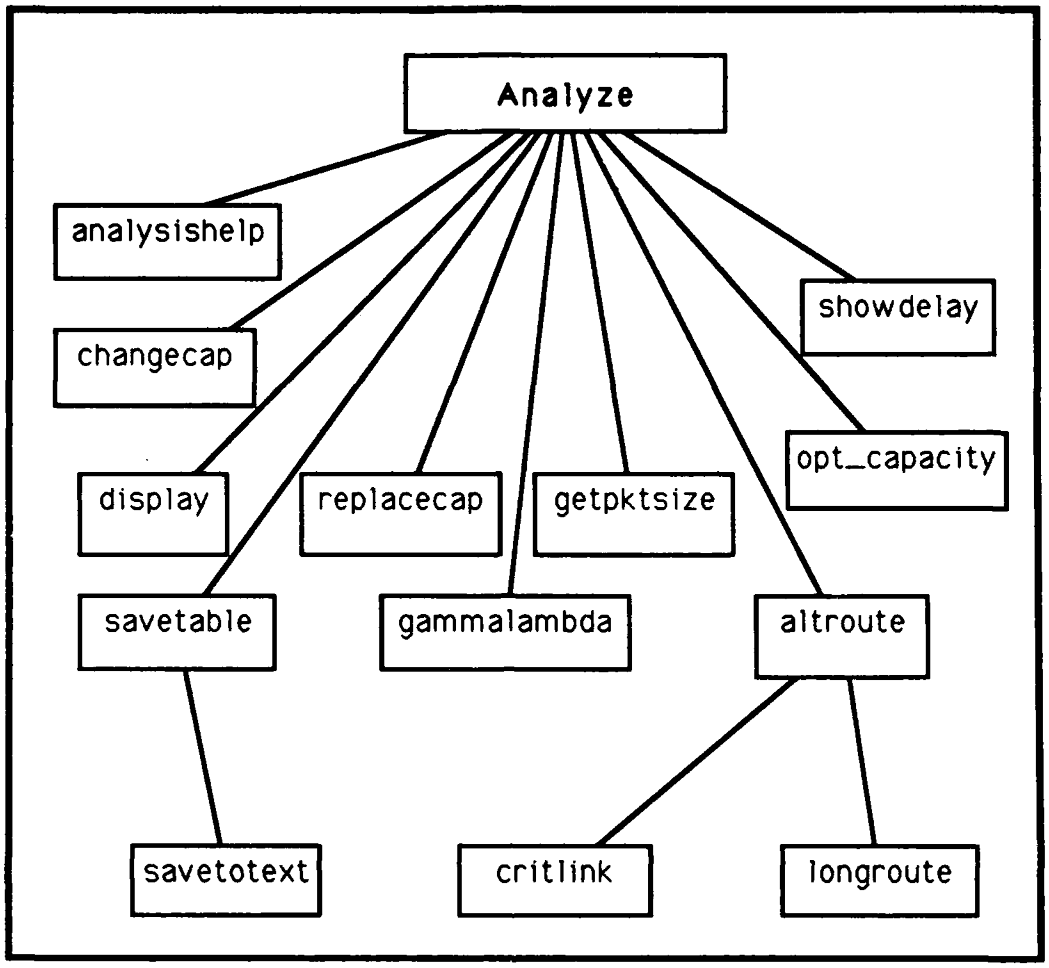


Figure 9. Structure chart for analyzer module

the source code available from the department of Electrical and Computer Engineering in the School of Engineering at AFIT for specific details.

Detailed Development.

Figures 6 through 9 depict the final structure of the application. Comparisons are now made with the original design of Figure 5.

Toplevel. Figure 6 shows a modification of the original design in terms of flow of control. Most notably the drawing and routing table loaders are called from the toplevel module in order to provide the flexibility of going directly to any of the three major modules (shown in boldface type). The analyze function is displayed as being subordinate to the drawnet and defroute functions but in actuality it can call either function under certain conditions (As explained in the User's Manual included in the Appendix).

The preanalyze function is simply a test for the existence of an entry for packet size. If a packet size has already been defined program control transfers directly to analyze. Otherwise it prompts for and receives user input for the parameter.

Drawing Editor. Figure 7 depicts the implementation of the drawnet module. It is left relatively intact with the exception of an additional module. "Quitroute" was added to check for updates in the routing table before control is transferred away from the drawing editor. It decides if changes have been made, saves the edited route if so instructed using "saveroute", and then returns control to the "drawnet" executive.

The "delete" module has two subfunctions that delete either a node or link. The "dellnks" function performs the actual mechanics of deleting a link and is used for both "delnode" and "dellink". It is called by both to provide the feature of having a node deletion automatically invoke the deletion of all links that are connected to it.

Also down in the hierarchy is the "refscrn" function. It is called by several higher level functions to redisplay the database after changes have been made, redrawing all of the network graphic components by cycling through the link and node data structures.

Routing Editor. Reference Figure 8. The routing editor keeps all functions defined in the structure design drawing and adds more detail. "Savedwg" is necessary in order to check for updates to the drawing before leaving the module. Also, the "refscrn" and "saveroute" functions are called from this module as they were from the "drawnet" module. The only new component is the "buildtable" function which creates and updates the display of the routing table in the background page of screen memory as routes are added or deleted. The memory resident quality is useful in that the routing table can be viewed virtually instantaneously from the routing editor by using the viewroute option.

Analyzer. Figure 9 shows the final structure chart for the third module. By comparison with Figure 5 one can see little modification from the original design. The one additional function called from the executive module is the "display" function which writes the results to the screen in table form. Subordinate to the "savetable" function is the "savetotext" module which creates an ASCII file copy of the display.

Subordinate to the "altroute" module are "critlink" and "longroute". "Critlink" searches for the link that has the greatest traffic intensity. "Longroute" searches through the route database and displays all routes that contain the link returned by "critlink". It also computes and displays the sum of all delays along each flagged route.

Testing

Black Box Testing. Black box testing was performed by first determining the equivalence classes and then devising test cases. The test cases covered both valid and invalid equivalence classes using random and boundary values for inputs. Appendix B lists the set of equivalence classes and test cases used.

Functional Testing. Functional testing was performed (a subset of exhaustive white box testing) during the iterative coding and testing process of implementation. New tests were performed with each addition of a logically coherent section of code.

Comparisons to Other Simulations. The example from the Tanenbaum text was modeled using both this simulator and COMNET II.5 from CACI (8). COMNET is a telecommunications network simulator that uses adaptive routing rather than static routing. It also does not assume independent links for end-to-end performance measures. Its results always include startup and steady-state values rather than steady-state only as in this project.

Field Testing and Evaluation. Members of an introductory class in computer networks tested and evaluated the software as applied to a networking problem. The software was judged mainly on the quality of the user interface and its effectiveness as a learning tool. Participants were asked to complete an evaluation form and make general comments.

Other Tests. The capability to handle large networking problems was tested by creating a network with 60 nodes and 106 links and then observing how the program handled node and link deletions as well as screen refreshes. Also tested was the consistency of the drawing elements as a function of their orientation on the screen. This test was performed by drawing links at approximately four degree intervals radiating from a central node in a full 360 degree sweep. Examples are included at the end of Appendix B.

V. Results and Discussion

Development Methodology

The application developed in this study was the result of a process of discovering capabilities and requirements and then applying the software development process to reach the solution. The reason behind any perceived success is attributed largely to modularity of design and quality of the development environment.

There were some aspects of the study that were more challenging than others. Getting started was a slow process due to the large selection of development environments available. Much time was spent writing and testing small drivers and routines that manipulated the color graphics and the mouse; however, once these low-level issues were resolved the network algorithms were implemented without a great deal of difficulty. The Turbo C compiler (the environment finally decided upon) seems to produce reasonably fast executable code without having to resort to assembly language routines. It is not known how well the product may have turned out if some other set of tools had been used.

As a side note, the entire application was created without the use of a "goto" statement; however, many of the functions were terminated by the use of another function call, which was often of a recursive nature. This method of coding provides the power and flexibility of the "goto" statement without creating the readability problems often associated with it.

Test Results

Black Box and Functional Tests. In the opinion of the author, testing probably serves its most useful purpose when modifications are done to an existing design. As mentioned earlier many requirements were determined and modified throughout the development phase of the study. As each set of requirement modifications were implemented, subsets of the suite of test cases presented in Appendix B were run. Most of the errors discovered were not of the classical software engineering type, that is to say, data and code corruption did not appear to any great extent. This result can be attributed to the use of a structured language along with modularity of the design.

The majority of errors that did appear had to do with graphics issues. For example, if a certain object on the screen were changed, another object created by some other function might be affected by being overwritten or crowded. These problems were easily spotted due to the instant feedback nature of the CRT display.

Comparison with COMNET II.5. The networking problem presented in the Tanenbaum text was modeled using both this simulator and the COMNET package (8)(11:63-64). Results of both methods are shown in Figure 10. Although the trends are similar the average delay shown by the COMNET curve is consistently lower. The reason for the different results is probably attributed to the routing stratagies used. COMNET implements a routing optimization technique where the routing selected in the example is somewhat random. Another notable difference in the two methods is COMNET's lack of use of the independence assumption. In other words, in COMNET a packet maintains a constant size from origin to destination rather than being repeatedly regenerated along its route.

It should be noted that COMNET can be considered to be more of a "true" simulator in that it produces entities and collects statistics. The software presented in this work is concerned more with the application of mathematical models to various network designs originated by the user and therefore does not fall within a strict definition of simulation software.

Field Test Results. Fifteen students in an introductory course in computer networks were given the task of applying the tool to a design problem and then evaluating the effectiveness of the software. An evaluation form was distributed to each student

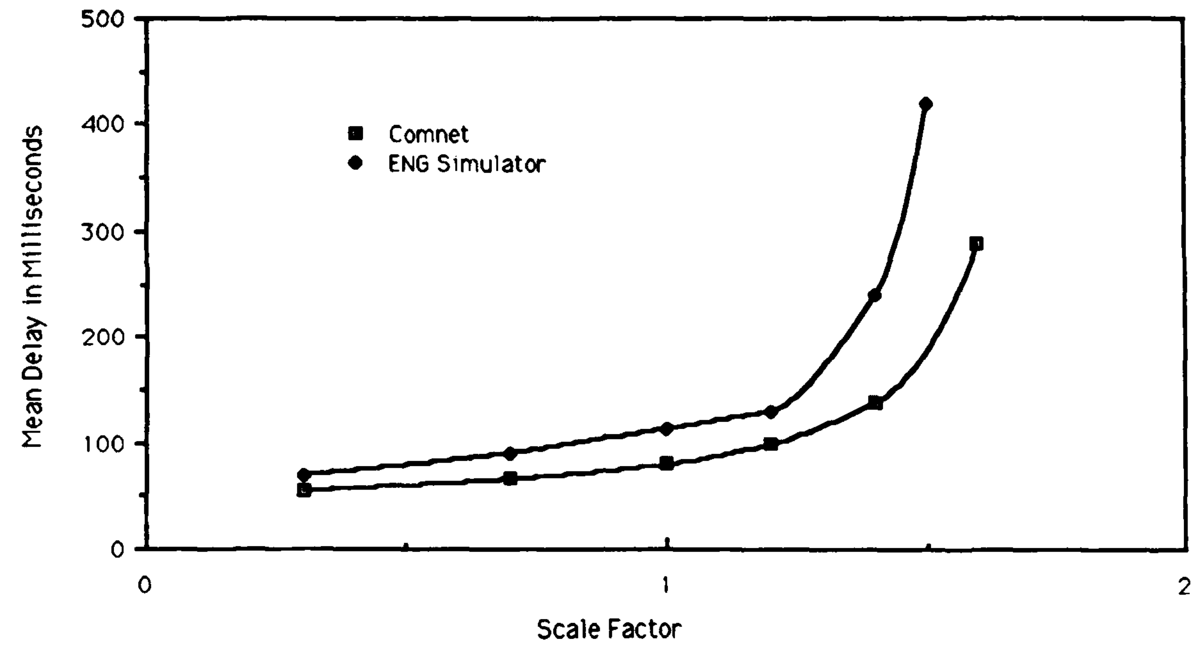


Figure 10: Comparison with COMNET II.5

along with the problem. The students were graded on how well the tool was applied to the exercise, thus encouraging a serious effort.

Results of the evaluation along with a sample of the form are shown in Appendix C. As a result of the evaluation the following observations are presented. In regard to quality and usefulness the tool got good marks. Almost all of the students felt that the application was easy to learn and use and should be offered as a learning tool to supplement the lecture series. The major problems were with compatibility issues, specifically, there was a wide range of responses concerning the requirement for EGA graphics and the mouse. Intuitively, the likelihood that one would have no objection to this requirement is related to what happens to be the configuration of the individual's personal computer system. Also, most students would rather use the software on their own systems rather than the Z-248's in the laboratory which are properly configured.

Another perception of deficiency in the tool was the lack of a capability of being able to copy files from within the program rather than having to exit, copy the file, and then return. This feature was simply overlooked in the software design and should have been included. Students also suggested that the application provide the capability to calculate total capacity of the network. This recommendation is probably more a result of the specific problem assigned rather than a general observation.

Recommendations

One of the goals as outlined in the thesis proposal was the possible inclusion of an expert system that could evaluate a specific network routing configuration and then make decisions as to alternate routing schemes. This feature was not completed due to the lack of success in implementing an effective and intelligent search algorithm. The attempts that were made at implementing various expert system techniques were met with limited success. The resulting functions were somewhat unreliable or inconsistent in execution and tended to degrade the overall quality of the program.

In the source code accompanying this paper a separate compilation unit entitled "altroute.c" has been reserved for possible future development. As of this writing the module has the capability to search through the data for the link with the highest intensity and then identify all routes that contain the link. In addition, the function will sum all of the values for the average delays of the links that make up each route identified . It will then display each route label along with its corresponding delay sum. The user is next prompted for a decision as to whether or not the routing table is to be modified. If the user responds in the affirmative control is transferred to the routing editor. The logical extension to this module would be to go back into the data base and find an alternate route for the one causing the largest degree of aggregate delay.

Users all seemed to be in agreement that the graphical interface was a strength of the tool; therefore, it is conceivable that this same interface be used for more sophisticated network models such as those which use dynamic routing or allow for operational overhead like propagation delay or error handling. The graphics source code (net.c) may be used with little or no modification for a variety of analytical techniques that could be included in the analysis module (analysis.c).

In the near term some other modifications are in order as a result of the in-class evaluation. Most notably is the file copy capability mentioned earlier. Also, a CGA version could be developed by using the set of CGA drivers already available in the Turbo C package (in lieu of the EGA/VGA routines) and then modifying the pixel coordinates to allow for the reduction in resolution.

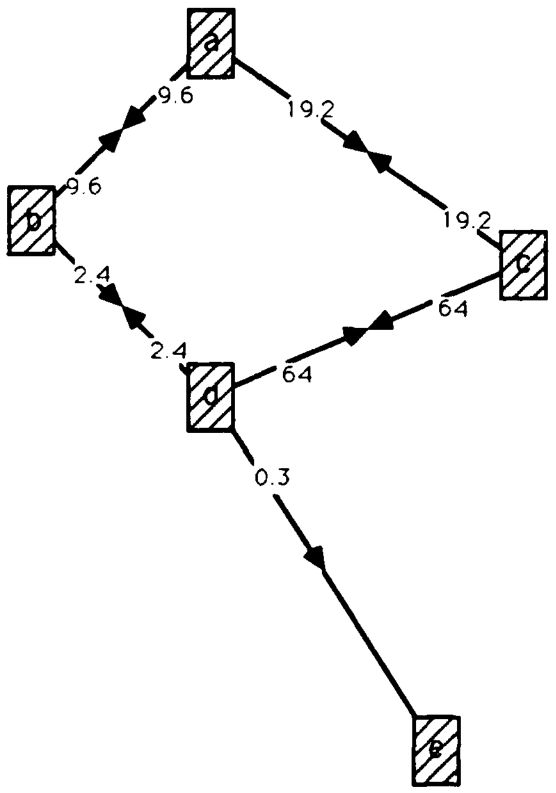
Appendix A:

User's Manual

User's Manual

AFIT/ENG COMPUTER NETWORK SIMULATOR

Version 1.0



by Ralph Puckett

December 1988

USER'S MANUAL for

AFIT/ENG COMPUTER NETWORK SIMULATOR

System Requirements	IBM Compatible PC
	EGA Monitor and Graphics card with 256K
	Microsoft Compatible Mouse
	IBM Graphics Compatible Printer

Overview

The simulator is composed of three main modules - a network network editor, a routing editor, and an analyzer. The first two require the EGA graphics and mouse, while the analyzer will run on just about any PC Compatible. The basic flow of the program is to first construct a network and assign link capacities. After a network has been defined the user calls up the routing editor to select routes and enter appropriate traffic levels. With the network and routes defined in the first two modules the user can assign a packet size and run an analysis. The analysis is based on an M/M/1 queue for packet arrivals. Results include traffic, average delay, optimal capacity assignment, and traffic intensity for each link as well as average delay for the network. Both the drawing and the analysis results can be saved and/or printed.

Three different disk files are created when drawing a network: ".nde", ".lnk", and ".rte", corresponding tc the sets of nodes, links, and routes. Also the analysis results can be saved in a ".txt" file for viewing and printing. A saved drawing must contain both nodes and links or the analyzer will not recognize it when an attempt is made to open it again. The routing table and analysis results are optional.

Theoretically, the simulator will allow networks of any size but as a practical matter the screen size will limit the amount of graphical data. The software has been tested with as many as 66 nodes and 120 links with no problems.

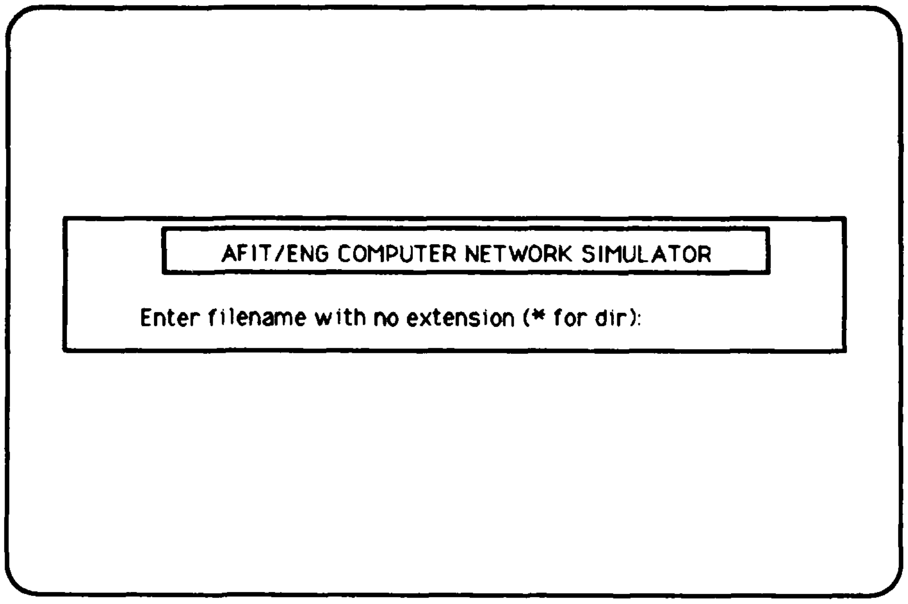
Getting Started

One of the basic ideas driving the design of this simulator was simplicity of use. A short tutorial which follows should be all that's necessary to get started.

First, make sure that the following three files are in the default directory (i.e. you should be in the same directory as the files or include the executable files' directory in a path statement):

NET.EXE - main program
EGADMP.COM - EGA printscreen routine
GONET.BAT - batch file that loads the above files.

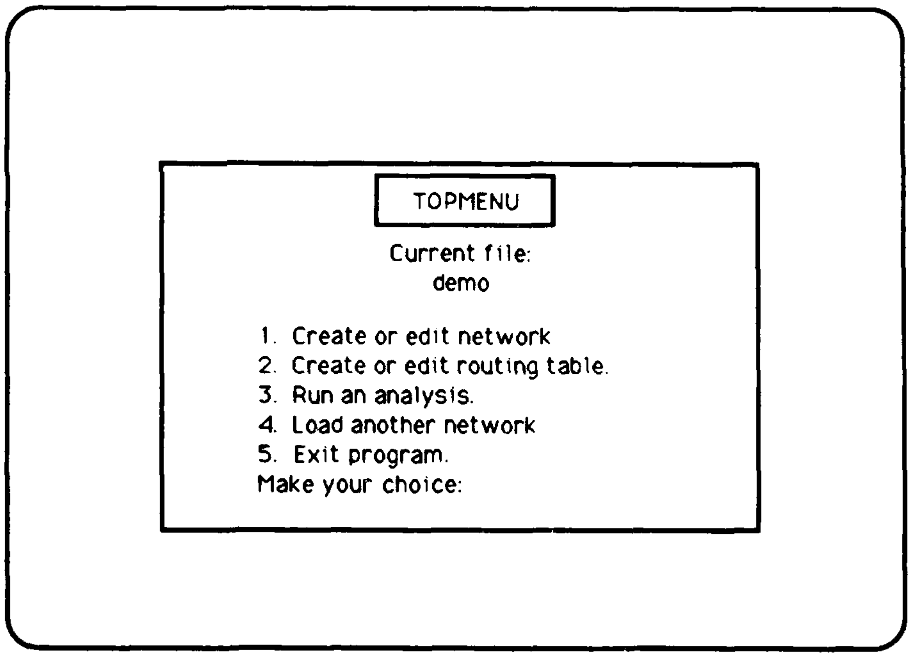
Enter "gonet". The screen print routine will load first followed by the main routine (Figure1). The program will then prompt for a filename, which may either be a new or existing file. If the file is not found, the program will create it. We will create a set of files named demo, so enter"demo" (without an extension). The next window (Figure 2) will display several choices. Since this is a new file enter '1' to create the network. The program will enter the graphics mode and enable the mouse (Figure 3). Displayed across the top are choices that are invoked by locating the arrow over a box and clicking any mouse button. To get started, mouse over to the upper left and click on the "Node" button, then locate the arrow near the upper left corner below the text and click. A rectangular node will be drawn followed by a prompt for a label. Type 'a'. The node will be labeled and the program will return you back to the top of the network editor.



```
AFIT/ENG COMPUTER NETWORK SIMULATOR

Enter filename with no extension (* for dir):
```

Figure 1: Opening Screen



```
TOPMENU

Current file:
demo

1. Create or edit network
2. Create or edit routing table.
3. Run an analysis.
4. Load another network
5. Exit program.
Make your choice:
```

Figure 2: Main menu

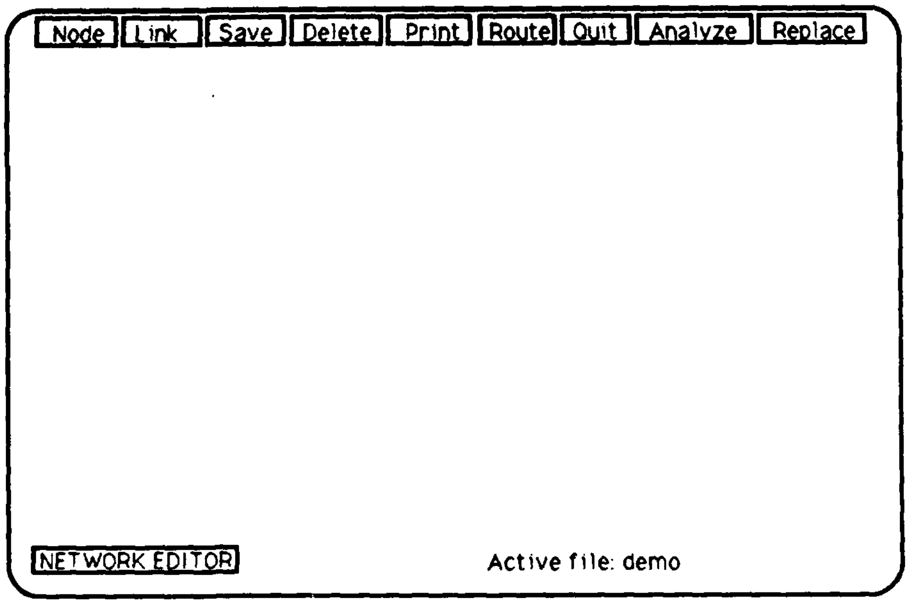
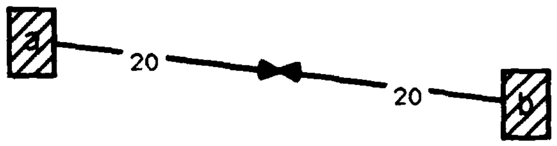


Figure 3: Network editor

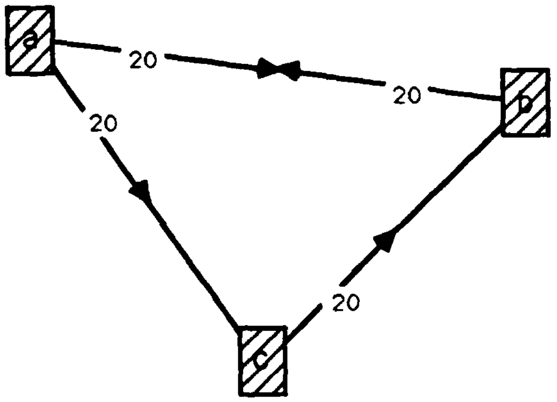
Now select "Node" again and place a second node about 2" to the right of node 'a' - label this one 'b'.

Now that we have two nodes we can go ahead and connect them with a link. Select the "Link" button and move the arrow anywhere over the 'a' node. Press and hold down a mouse button. With the button still pressed, move the arrow anywhere over node 'b' and release (this is sometimes referred to as "dragging"). After releasing the button you should see a line appear between the two nodes and a prompt on the topline of the screen asking for the link capacity. Enter "20". The link will be labeled with arrows in both directions and capacities in kbps for each direction. Your drawing should be similar to the following sketch:



A network is constructed using the above techniques to place nodes and connect them with links. For this exercise, we'll add one more node and connect it to the others with simplex links.

Select the "Node" button and place another node 'c' below the first two. Next construct a 20 kbps link from node 'a' to 'c' using the 'S' suffix when entering the capacity assignment. Do the same for the link from 'c' to 'b'. The final product should resemble the one in the figure.



Having constructed our sample network we're ready to go on to the routing editor. Mouse over to the "Save" button at the top and click, then do the same for the "Route" button. The program will now put you into the routing editor with its green menu buttons (Figure 4). Let's say we want to create a route from node'a' to node 'b' using links "ac" and "cb". Select the "Create a Route" button on the upper left. To create the route click on the nodes along the route in order, in this case, a-c-b. The nodes and links will be highlighted as they are selected. After clicking on node 'b' mouse up to the large "Stop" bar and click. You'll be asked to enter forward traffic in PACKETS PER SECOND rather than kbps. Enter '2'. The editor will next ask you for reverse traffic which should be none, so simply press the enter key. The route is now defined; to see the routing table click on the "View" button. Our simple routing table will be displayed showing the route and its traffic. Click any button to return to the editor.

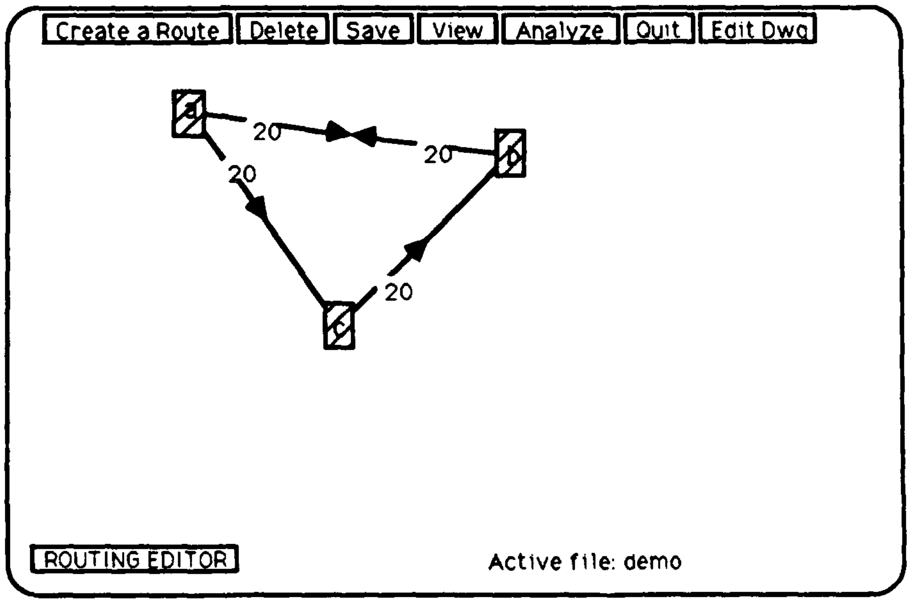


Figure 4: Routing editor

Next click on the "Analyze" button to perform an analysis of the newly created network. The routing editor will prompt you to save the routing table. Default is "yes" so hit any key except 'n'. The analyzer will now load and prompt for packet size – we'll use 800. Note that the packet size is entered in BITS. After entering the packet size the program will display the results of the analysis. Along the bottom is displayed a series of commands which are explained by typing 'H'. Note that the optimal cap column displays all zeroes until the user elects to calculate the optimal capacity assignments based on desired delay. Your results should be the same as those shown here.

```
         Arr Rate  Capacity  Capacity  Delay     Traffic   Optimal
1  Line  (pkts/s)  (kbps)    (pkts/s)  (ms)      Intensity Cap(kbps)
1   ac     2.00      20       25.00    43.478    0.08000    0.00
2   cb     2.00      20       25.00    43.478    0.08000    0.00
```

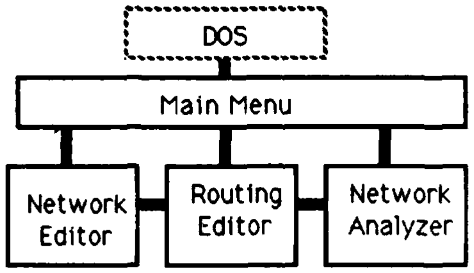
To exit the analyzer type 'q' and answer the prompts. Note that any changes to capacity assignments will update the .lnk files and be automatically reflected in the drawing if you elect to save the table. If you wish to save the results of the analysis for later viewing or printing, type 'y' at the "Write table to text file?" prompt.

This completes the tutorial.

Reference Section

Program Structure

The analyzer has one executive and three functional elements. The network analysis may be interrupted at any stage and any work in progress may be saved.



The network editor is where the model is originated and where the original capacity assignments for all links are made. Any addition or removal of a node or link must be done here. After a network has been drawn and all capacities assigned, the routing editor is selected from the row of selection buttons at the top of the network editor screen. The user may move freely back and forth between the network editor and routing editor.

The routing editor builds and displays the routing table. Routes are determined by using the mouse to select the nodes that define the route. The current routing table can be viewed quickly by selecting the "VIEW" command. (The routing table screen stays resident in memory).

Finally, the analysis module uses the sets of nodes, links, and routes to calculate the parameters of the network. The module provides on-screen help to explain the various commands. All of the parameters are displayed in a table that can be saved to a text file if desired.

The rest of this manual will explain the various commands and options available in each of the three modules - network (drawing) editor, routing editor, and analyzer, as well as the opening menu. (The terms Drawing Editor and Network Editor are interchangeable).

Top Menu

Displays choices. To begin building a network or to edit an existing drawing, type "1" followed by <Enter>. To create or edit a routing table type "2" followed by <Enter>. To run an analysis type "3 <Enter>". Note that routing and analysis are impossible without first creating a network using the network editor.

Network Editor

Node

Creates and labels a node. Node label must be a single, unique, printable character. Action may be canceled by clicking on the "Cancel" button at the top of the screen. Note that the program restricts the mouse movement such that a node can be placed only where it will be totally displayed. A node appears on the screen as a rectancle filled with blue slashes. User is prompted for a label which is displayed in the center of the node. Control is then returned to the network editor main menu. The collection of nodes is saved as a .nde file.

Link	Creates and assigns capacity in kbps to a link. Link is created by locating arrow over the node of origin and then clicking and dragging the mouse over to the destination node where the mouse button is released. Capacity assignment should be a numerical value with full-duplex being the default condition. A simplex node is designated by the addition of an 'S' suffix to the capacity entry. Only one link in either direction is allowed. All links for a particular network are saved as a .lnk file.
Save	Saves the drawing as .nde and .lnk files. If you decide to quit but forget to save the drawing the editor will ask whether or not to save changes (if editing was done).
Delete	Use to delete a network element. When selected the editor displays two choices corresponding to either deleting a node or deleting a link. Both are discussed below:
Delete a node	Point to the node to delete and click. All connecting links will automatically be eliminated. If an attempt is made to delete a node that is used by a route the program will disallow the action and inform you that a route is using that node. Before deleting the node that is in use you must first use the routing editor to delete all routes that use the node. To view the current list of routes, select first the "ROUTE" button from the network editor and once in the routing editor, select "VIEW".

Delete a link

Type the labels of the terminating nodes in proper order. If the link is not presently in use by a route, the editor will delete the link connecting the two nodes. For example, to delete the link connecting nodes 'a' and 'b', type "ab". No <Enter> keypress is needed.

If an attempt is made to delete a link that is used in the routing table the program will disallow the action and alert the user that the link is in use. All routes that use the link must first be deleted with the routing editor before the link can be deleted.

Note: The program will not delete a link or node that has been included in a route - the corresponding route must first be deleted using the routing editor.

Print

Prints the displayed network. When selected, the routine deletes the top menu and waits for a <Shift><Prtsc> key combination. For high quality printout, type 'h' immediately after <Shift><Prtsc>. It may be necessary to use the 12 CPI (elite) setting on printers that are not 100% IBM graphics compatible. Pressing the <ESCAPE> key will abort the print.

This option requires that the EGADMP.COM program be loaded and resident. To print analysis results, use the Save command from the Analyzer module to save the results as a text file. The text file can be printed using the DOS "PRINT <filename>" or "COPY <filename> PRN" command.

Route	Invokes the routing editor. The routing editor uses the drawing created by the network editor to develop a routing table. It is readily distinguishable by the green menu buttons. Changes made in the network editor will be retained and may be saved to disk later if desired.
Quit	Quits to the Topmenu. If changes have been made either to the routing or the drawing, user will be queried whether or not to save them. All changes may be discarded with a negative response.
Analyze	Invokes the analyzer. Program will ask if changes made are to be saved. Default is yes. Also prompts for packet size if none has been entered. The packet size is a mean value (exponentially distributed) in bits.
Replace	Allows deletion and construction of a link without the intermediate return to the top menu of the editor. Comes in handy during network modifications.

Routing Editor

Create a Route	Creates a route by successive selection of nodes along the route, beginning with the node of origin. As nodes are selected with the mouse, the network elements are highlighted in yellow to provide a trail. User is prompted for forward and return traffic in packets per second. Clicking on the "Stop" bar terminates selection of nodes or aborts if no nodes have been selected. Route is automatically added to the routing table and may be observed by selecting the "VIEW" button.

	A route may be defined over one or more links that do not exist. The analysis portion of the application will catch this and warn of the errors, after which it will automatically place the user in the network editor to add the missing links if desired.
Delete	Shows the current routing table and prompts for entry of a route to delete. If there is no match, no action will be taken.
Save	Saves the routing table as .rte file.
View	Displays the current routing table. Will display up to 81 routes. This display remains resident in memory and is quickly called up. This makes the routing and verification process smooth and less frustrating than if the screen were redrawn each time.
Analyze	Invokes the analyzer if there are no disagreements between the routing table and network. Will prompt user for packet size if value has not already been entered. The packet size is entered as bits per packet. If the routing table and drawing disagree the analyzer will sound the beeper and list the routes that are missing data links.
Quit	Quits to Topmenu. Will prompt for saving of any edits to the routing table or the drawing.
Edit Dwg	Invokes the network editor. The network and routing editors are freely switched from one to the other.

Analyzer	The commands for the analyzer are displayed at the bottom of the screen. Help can be displayed by typing 'h'.
P	Displays current packetsize in BITS. New packet size can be entered if desired. Typing only <Enter> puts you back in the analyzer without modifying the packet size. Changing the packet size will automatically cause recalculation of the network parameters.
D	Views the average delay for the entire network in milliseconds. This value can later be modified as a delay constraint for the network in order to find optimal capacity assignment (Refer to "O" command).
C	Enables the revision of any link capacity shown in the analysis table. If the analysis results are saved, the new capacity(s) will be automatically entered into the drawing. Typing <Enter> twice will abort the command and will not affect the analysis.
O	Calculate optimal capacity assignment based on desired delay in milliseconds. The optimal capacity assignments will be displayed in the right-most column. Optimal capacity assignments are zero by default until a delay constraint is entered using this command.
R	Replaces ALL capacity assignments with the current values shown in the Optimal Cap(kbps) column. If the current analysis is saved, the drawing will be updated to show all of the new capacities; therefore, use with caution. It's probably a good idea to keep a backup copy of any network drawing before experimenting with it.

A	Alternate routing. The critical link will be displayed along with the existing routes that contain the link. It also prompts for modification of the routing table. If 'y' is typed, the routing editor is invoked.
S	Save the results of the current analysis. The drawing will be updated to reflect any change in capacity assignments. Also prompts for storage of the results as a text file. The text file can be printed using the "PRINT <filename>" or "COPY <filename> PRN" commands from DOS or may be copied into a word processor.
Q	Quit to Topmenu.

Formulas:

γ_i : traffic for route i (packets/sec)

$\frac{1}{\mu}$: average packet size (bits/pkt) for network

C_i: link capacity in kpbs (kilobits per second)

μC : service rate (packets/sec)

λ_i : average arrival rate (packets/sec) for link i (Sum of all traffic γ using the link)

d : cost per kbps (Normalized to 1).

$T_i = \frac{1}{\mu C_i - \lambda_i}$: average delay for link i

$\gamma = \sum_{i=1}^{m} \gamma_i$: sum of all traffic on the net

$\lambda = \sum_{i=1}^{m} \lambda_i$: sum of all link arrival rates

$\rho_i = \frac{\lambda_i}{\mu C_i}$: traffic intensity for link i

$T = \frac{\lambda}{\gamma}$: mean packet delay for network

$$C_i = \frac{\lambda_i}{\mu}\left(1 + \frac{1}{\gamma T}\frac{\sum_{j=1}^{m}\sqrt{\lambda_j d_j}}{\sqrt{\lambda_i d_i}}\right)$$: optimal capacity for link i based on delay constraint T

Appendix B:

Tests

Appendix B

Equivalence Classes

External Condition	Valid Equivalence Classes	Invalid Equivalence Classes
Opening menu: Filename	1 - 8 Characters(1), No file extension(3) <Enter>(5)	>8 Characters(2) File extension(4)
Viewdir: Filename	1-8 Characters(6) No file extension(9)	<1 Character(7), >9 Char(8) File extension(10)
Topmenu: Menu selection	'1'(11), '2'(12), '3'(13), '4'(15), or '5'(16)	Not in {1 .. 5} (14)

Drawing Editor Module

External Condition	Valid Equivalence Classes	Invalid Equivalence Classes
Drawnet: Mouse button pressed	Active region(17) ("Hotspot")	Inactive region(18)
Drawnode: Mouse down	Drawing area(19) Cancel(21)	Outside drawing area(20) Keypress(22)
Drawnode: Nodelabel	Unique printable character(24)	Non-printing character(23) Non-unique character(25)
Drawlink: Draw a link	Cancel(26) Connect two nodes w/o overwriting existing link(29)	Not connecting 2 nodes(27) Overwrite existing link(28)

Drawlink: Linkcapacity	1 - 6 characters FDX(30) 1 - 6 characters SPX(32)	> 6 Characters(31) < 1 Character(33) FDX over existing SPX(34) Non-printing Characters(35)
Save Command	No editing done(36) Save with changes(38)	Null database(37)
Delete Command: Delete a node	Select node(39) Cancel(41)	Keypress(40) Select blank space(42)
Delete Command: Delete a link	Displayed link symbol(43)	< 2 Characters(44) Mouse down(45) Non-existent link(46)
Print	<Shift><Prtsc>(47) <Shift><Prtsc> & 'h'(49) Any other key(50)	Mouse down(48)
Quit: Save drawing prompt	'y'(51), 'n'(52),Not 'y' or 'n'(53)	
Replace	Valid link symbol(54) <Esc>(56) Same as drawlink(57)	Arbitrary string(55) Other non-printing character(58)

Routing Editor Module

External Condition	Valid Equivalence Classes	Invalid Equivalence Classes
Routing editor	Valid Selection(59)	Keypress(60) Invalid selection(61)

Create a Route	Sequence of nodes(62) Stop selection(64)	Selections outside nodes(63) Keypress(65)
Delete	<Enter>(66) Valid route symbol(68)	Other non-print char(67) Nonexistent route(69)
Save	Changes(70)	Null routing database(71) No changes made(72)
View	View routing table(72a)	
Quit: saves changes?	y(73) , n(74),Not y or n(75)	

Analysis Module

External Condition	Valid Equivalence Classes	Invalid Equivalence Classes
Analyze: Bits per packet	Pos floating point number(77)	Non positive floating pt(76) Non-numeric entry(78)
Analyze: menu selection	Displayed selections(79)	Non-displayed char (80)
Pktsize window prompt	Pos floating point(81) <Enter>(83)	Non positive floating pt(82) Non-numeric entry(84)
Changecap:		
Linklabel	Displayed link(85)	Arbitrary string(86)
Capacity Assignment	1 - 6 digits(87)	<1(88), >6 digits(89), Non-numeric(90)
Optimal capacity: delay	>0 and <999999(92)	<0(93), >999999(94) Non-numerical(95)
Replace capacities with opt	y(96) , n(97)	Not y or n(98)

Alternate routing	y(99) , n(100),Not y or n(101)
Save: Write to text file	y (102), n(103),Not y or n(104)
Quit: Save table?	y (105), n(106),Not y or n(107)
Help	Any key(108)

Test Cases

Toplevel Routines

Test Case ID	Test Input	Equivalence Classes Covered
A	"test"	1, 3
B	<Enter>	5
C	"test_test"	2
D	"test.rte"	4
E	"test"	6, 9
F	<Enter>	7
G	"test_test"	8
H	"test.rte"	10
I	1	11
J	2	12
K	3	13
L	4	15
M	5	16
N	X	14

Drawing editor

Test Case ID	Test Input	Equivalence Classes Covered
A-a	Drawnode: Cancel	17, 19, 21, 24, 26, 29, 30, 32, 36, 38,
-b	Drawnode: valid area	39, 41, 43, 47, 49, 50, 54, 56, 57, 51
-c	Nodelabel = 'a'	
-d	Create nodes 'b' and 'c'	
-e	Drawlink: Cancel	
-f	Drawlink: connect nodes 'a' and 'b'	
-g	Capacity = "20"	
-h	Drawlink: connect nodes 'b' and 'c'	
-i	Capacity = "20s"	
-j	Save with changes	
-k	Save, no changes	
-l	Delete node: cancel	
-m	Delete node: valid node	
	-n	Delete link: valid label
	-o	Print: <Shift><Prtsc>
	-p	Print: <Shift><Prtsc>'h'
-q	Print: Any other key	
-r	Replace: valid linksymbol	
-s	Replace: Draw link	
-t	Replace: <Esc>	
-u	Save drawing: 'y'	
B	Quit without saving	52
C	Drawnet: Inactive region	18
D	Drawnode: Check mouse boundaries	20
E	Drawnode: press any key	22
F	Nodelabel: <BS>	23

G	Nodelabel: Repeat label	25
H	Drawlink: Node to itself	27
I	Drawlink: Space to Node	27
J	Drawlink: Node to space	27
K	Drawlink: Space to space	27
L	Drawlink: Duplicate link	28
M	Linkcapacity: "2000000"	31
N	Linkcapacity: <Enter>	33
O 34	Linkcapacity: FDX over SPX	
P	Linkcapacity: <BS>	35
Q	Save: No data	37
R	Delete: Press any key	40
S	Delete: Click outside selection	42
T 44	Delete link: 'a' and <Enter>	
U	Delete link: Mouse click	45
V	Delete link: "xy"	46
W	Print: Mouse click	48
X	Quit: <BS>	53
Y	Replace: "xy"	55

Z	Replace: <BS>	58

Routing Editor

Test Case ID	Test Input	Equivalence Classes Covered
A-a	Select "Create a Route"	59, 62, 64, 66, 68, 70, 72a, 74
-b	Select 'a' -> 'b'	
-c	Stop selection	
-d	Delete: <Enter>	
-e	Delete: "ab"	
-f	Save	
-g	View: Mouse click	
-h	Quit: 'n'	
B	Quit: Edit and quit, 'y'	73
C 75	Quit: Edit and quit, <BS>	
D	Topmenu: Press any key	60
E	Topemenu: Click no selection	61
F 63	Create: Click outside nodes	
G	Create: Press any key	65
H	Delete: <BS>	67
I	Delete: "xyz"	69
J	Save: No routes	71

K	Save: No changes	72

Analysis Module

Test Case ID	Test Input	Equivalence Classes Covered
A	1000	77
B	-1000	76
C	abcdefg	78
D	P	79
E	X	80
F	1000	81
G	0	82
H	<Enter>	83
I	<BS>	84
J	"ab"	85
K	"xyz"	86
L	1	87
M	-1	88
N	20000000	89
O	"abcdef"	90
P	5000	92

Q	-100	93
R	1000000	94
S	"~!@#$%^"	95
T	Y	96
U	N	97
V	<Enter>	98
W	y	99, 102, 105
X	n	100, 103, 106
Y	<Enter>	101, 104, 107
Z	<Enter>	108

Test Cases: Boundary Values

Toplevel Routines

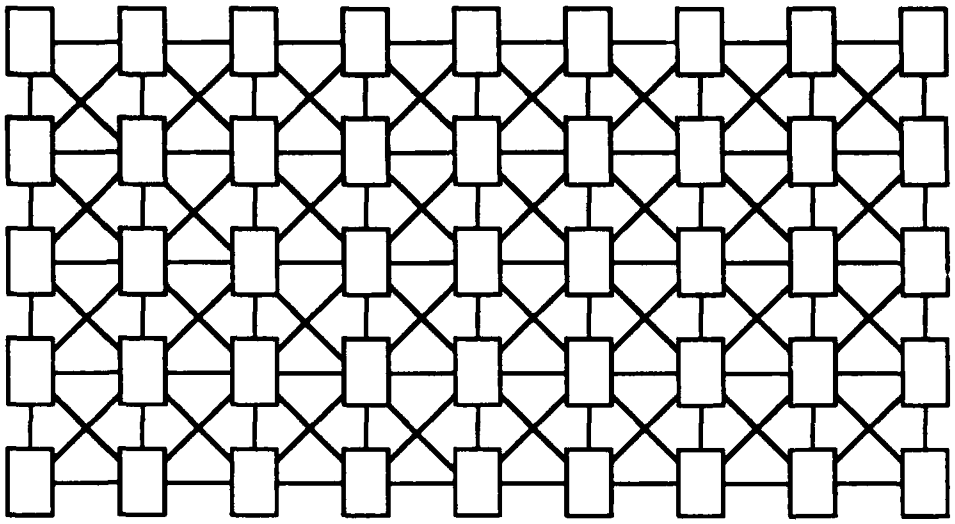
Test Case ID	Test Input	Equivalence Classes Covered
A	"a"	1
B	"testname"	1
C	"alongname"	2
D	<Enter>	5
E	"a"	6

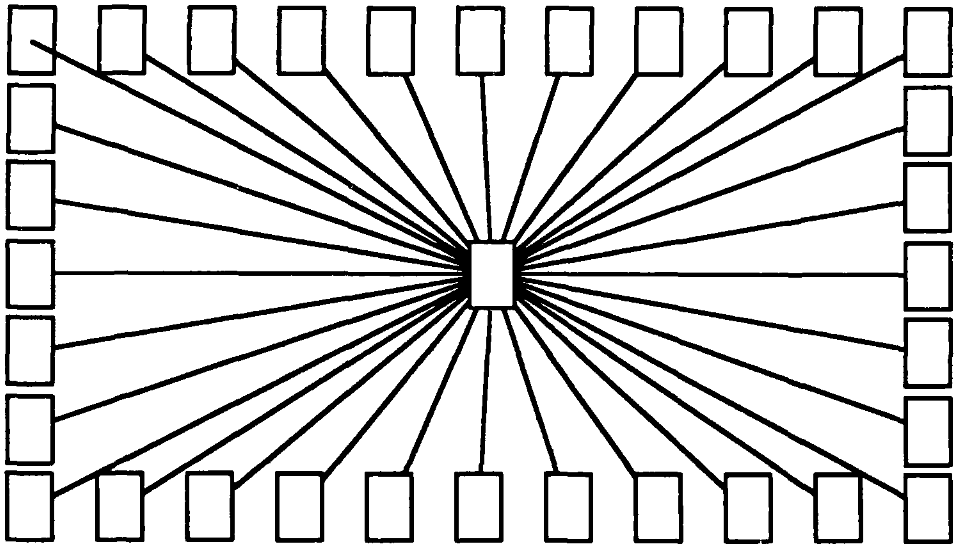
F	"testname"	6
G	<Enter>	7
H	"alongname"	8
I	'1'	11
J	'5'	16
K	'0'	14
L	'6'	14

Other Routines

Test Case ID	Test Input	Equivalence Classes Covered
A	"1"	30
B	"100000"	30
C	"1000000"	31
D	"1s"	34
E	"100000s"	34
F	<Enter>	33
G	1	77
H	0	76
I	-9999999	78
J	1	87

K	999999	87
L	0	88
M	1000000	89
N	.000001	91
O	1000000	94





Multinode and arrow tests

Appendix C: Software Evaluation Survey Results

This survey will be used to evaluate the operation of the AFIT/ENG Computer Network Simulator. Please choose one of the five possibilities and feel free to add comments below your response. There are some questions at the end and some space for any additional comments. Thanks for participating.

1	2	3	4	5
Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

1. Most CAD packages that I've had experience with are somewhat difficult or awkward to learn and use.

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

Mean = 3.27; Std Dev = 0.704; Low = 2; High = 4.

2. The mouse is better suited as a drawing device than the keyboard.

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

Mean = 4.73; Std Dev = 0.458; Low = 4; High = 5.

3. For most CAD programs that I've used in the past but don't use all the time, "relearning" the commands and program flow takes too long.

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

Mean = 3.13; Std Dev = 0.834; Low = 2; High = 5.

4. Getting started. This network simulator has a reasonably short learning curve.

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

Mean = 4.87; Std Dev = 0.352; Low = 4; High = 5.

5. On-screen help. The amount and quality of on-screen help for the simulator is adequate.

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

Mean = 4.33; Std Dev = 0.617; Low = 3; High = 5.

6. User friendliness Overall the simulator has a somewhat intuitive interface.

☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

Mean = 4.40; Std Dev = 0.632; Low = 3; High = 5.

7. Speed. The simulator seems to execute all of its commands quickly enough.

☐	☐	☐	☐	☐
1	2	3	4	5

Mean = 4.53; Std Dev = 0.834; Low = 2; High = 5.

8. Portability. The requirements for EGA and a mouse are reasonable.

☐	☐	☐	☐	☐
1	2	3	4	5

Mean = 3.00; Std Dev = 1.13; Low = 1; High = 5.

9. Usefulness. The simulator helped me understand the operation and performance characteristics of a packet-switched network.

☐	☐	☐	☐	☐
1	2	3	4	5

Mean = 4.20; Std Dev = 0.561; Low = 3; High = 5.

10. This software can be of value to an introductory course in computer networks.

☐	☐	☐	☐	☐
1	2	3	4	5

Mean = 4.47; Std Dev = 0.640; Low = 3; High = 5.

Questions:

1. What improvements do you think are needed? What additional feature(s) should be added?
- **CGA Capability (4 responses)**
- **Keyboard only capability (2)**
- **Ability to save data to another file without leaving program (7)**
- **Ability to calculate total network capacity (2)**

2. Did you uncover any major bug(s) in the software? If so, what?
- **Incapability with a certain brand of third party graphics board with enhanced EGA - 640 by 480 pixels (1)**
- **Problems when deleting all elements in a file - probably got a null pointer assignment (1)**

3. Any general comments?
- **Mostly positive. Negative comments were extensions of responses given in Question #1 above.**

Bibliography

1. Adams, Charles A. Jr. A Digital Circuit Design Environment, MS Thesis AFIT/ENG/GCS/87D-1. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base OH, December 1987.

2. Booch, Grady. Software Engineering with Ada (Second Edition). Menlo Park, California: The Benjamin/Cummings Publishing Company, Inc., 1987.

3. Carroll, John M. and Sandra A. Mazur. "LisaLearning", IEEE Computer. 19: 35-49. (November 1986).

4. Hayes, Jeremiah F. Modeling and Analysis of Computer Communications Networks. New York: Plenum Press, 1984.

5. Howden, William E. "A Functional Approach to Program Testing and Analysis", IEEE Transactions on Software Engineering, SE-12: 997-1005. (October 1986).

6. Johnson, Nelson. Advanced Graphics in C, Programming and Techniques. Berkeley, California: Osborne McGraw-Hill, 1987.

7. Kleinrock, Leonard. Queueing Systems, VOL. 1: Theory. New York: John Wiley, 1975.

8. Pracher, John. "Simulation of Network Routing Strategies", Special Project Report, School of Engineering, Air Force Institute of Technology, September, 1988.

9. Pressman, Roger S. Software Engineering, A Practitioner's Approach (Second Edition). New York: McGraw-Hill Book Company, 1987.

10. Schildt, Herbert. Turbo C: The Complete Reference. Berkeley: Osborne McGraw-Hill, 1988.

11. Tanenbaum, Andrew, S. Computer Networks. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1981.

VITA

Captain Ralph Puckett [redacted] He attended North Carolina State University at Raleigh where he graduated with a degree in Zoology. His first position after graduation was laboratory research technician for the U.S. Environmental Protection Agency where he co-authored two published papers on the chemical analysis of environmental pollutants.

Capt Puckett received his commission from the Officer Training School in 1982 and was subsequently assigned to an AFIT sponsored tour where he received his B.S. in Electrical Engineering from Louisiana Tech University. After graduation from Tech he went through the Air Force Communications Command comm-electronics officers course at Keesler AFB, Mississippi.

In May of 1984 Capt Puckett was assigned to OLAA, 6008 Tactical Air Control Flight (PACAF), Wheeler AFB, Hawaii. He served as Deputy Chief, Engineering and Installation and was responsible for overseeing projects associated with Project Constant Watch - an upgrade to the command and control system in support of U. S. Air Force tactical operations in the Republic of Korea.

Capt Puckett received his Master's Degree in Business Administration from Chaminade University in Honolulu in December 1986 . He entered the School of Engineering at AFIT in May 1987.

[redacted]

[redacted]

[redacted]

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; Distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/ENG/88D-39	5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION School of Engineering	6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, OH 45433-6583		7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS (City, State, and ZIP Code)

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
DESIGN AND IMPLEMENTATION OF A PC BASED NETWORK SIMULATOR FOR ENGINEERING EDUCATION

12. PERSONAL AUTHOR(S)
Ralph D. Puckett, Captain, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM ______ TO ______	1988 December	82

16. SUPPLEMENTARY NOTATION

fr. back

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP
12	05	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Computer Graphics, Computer Aided Design, Computer Networks, Theses. (SDW)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Thesis Chairman: Bruce L. George, Captain, USAF
Assistant Professor of Electrical Engineering

Approved for release in Accordance with AFR 190-1
[illegible]
10 Jan 89

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Bruce L. George, Captain, USAF	22b. TELEPHONE (Include Area Code) (513) 255-2024	22c. OFFICE SYMBOL AFIT/ENG

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

19 (cont)

The purpose of this study was to design and implement a simulator to assist students of computer networks. The basic objective was to create a software application that provides rapid feedback on network design decisions. Of particular interest is the packet switched network with data links of various capacity assignments. Another basic objective was to create a graphics interface that eliminated the need to learn a simulation language while still maintaining a powerful and useful product.
The end product was a result of the application of both networking theory as well as software engineering principles with particular attention being paid to reliability and maintainability. With this tool the student can create any network topology simply by pointing and clicking a mouse and entering a few network parameters from the keyboard. The application can be run on a personal computer - an environment which is accessible and fairly well understood. Keywords:
The design and implementation of the application is presented in this work along with the results of developmental testing. In addition, a sample of fifteen students was chosen to provide the initial beta test and those results are presented. A comprehensive user's manual is included as an appendix. Finally, several recommendations concerning future development are discussed.

FLD 18